Hot Topics

LNCS 2924

Jamie Callan
Fabio Crestani
Mark Sanderson (Eds.)

Distributed Multimedia Information Retrieval

SIGIR 2003 Workshop on Distributed Information Retrieval
Toronto, Canada, August 2003
Revised Selected and Invited Papers

Springer

Lecture Notes in Computer Science 2924
Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Hong Kong
London
Milan
Paris
Tokyo

Jamie Callan Fabio Crestani
Mark Sanderson (Eds.)

Distributed Multimedia Information Retrieval

SIGIR 2003 Workshop on Distributed Information Retrieval
Toronto, Canada, August 1, 2003
Revised Selected and Invited Papers

Springer

Volume Editors

Jamie Callan
Carnegie Mellon University, School of Computer Science (LTI)
5000 Forbes Avenue, 4502 Newell Simon Hall
Pittsburgh, PA 15213-8213, USA
E-mail: callan@cmu.edu

Fabio Crestani
University of Strathclyde, Department of Computer and Information Sciences
Livingstone Tower, 26 Richmond Street, Glasgow G1 1XH, UK
E-mail: f.crestani@cis.strath.ac.uk

Mark Sanderson
University of Sheffield, Department of Information Studies
Regent Court, 211 Portobello St, Sheffield, S1 4DP, UK
E-mail: m.sanderson@shef.ac.uk

Cataloging-in-Publication Data applied for

A catalog record for this book is available from the Library of Congress.

Bibliographic information published by Die Deutsche Bibliothek
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data is available in the Internet at <http://dnb.ddb.de>.

CR Subject Classification (1998): H.3, H.4, H.2, C.2.4, H.5.1

ISSN 0302-9743
ISBN 3-540-20875-5 Springer-Verlag Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by PTP-Berlin, Protago-TeX-Production GmbH
Printed on acid-free paper SPIN: 10978702 06/3142 5 4 3 2 1 0

Preface

During the last decade companies, governments, and research groups worldwide have directed significant effort towards the creation of sophisticated digital libraries across a variety of disciplines. As digital libraries proliferate, in a variety of media, and from a variety of sources, problems of resource selection and data fusion become major obstacles. Traditional search engines, even very large systems such as Google, are unable to provide access to the "Hidden Web" of information that is only available via digital library search interfaces. Effective, reliable information retrieval also requires the ability to pose multimedia queries across many digital libraries. The answer to a query about the lyrics to a folk song might be text or an audio recording, but few systems today could deliver both data types in response to a single, simple query. Distributed information retrieval addresses issues that arise when people have routine access to thousands of multimedia digital libraries.

The SIGIR 2003 Workshop on Distributed Information Retrieval was held on August 1, 2003, at the University of Toronto, following the SIGIR conference, to provide a venue for the presentation and discussion of recent research on the design and implementation of methods and tools for resource discovery, resource description, resource selection, data fusion, and user interaction. About 25 people attended, including representatives from university and industrial research labs. Participants were encouraged to ask questions during and after presentations, which they did. The formal presentations were followed by a general discussion of the state-of-the-art in distributed information retrieval, with a particular emphasis on what still needs to be done.

This volume includes extended and revised versions of the papers presented in the SIGIR 2003 Workshop in addition to a few invited papers.

Overview

The book is structured into four parts covering the major areas of research associated with multimedia distributed information retrieval: resource discovery, resource selection, data fusion, and architectures. Below we provide very brief descriptions of the papers included in the volume, to give a sense of the range of themes and topics covered.

Harvesting: broadening the field of distributed information retrieval by Fox et al. argues that in addition to federated search and gathering, harvesting is an important approach to address the needs of distributed information retrieval. The paper is centered on the user of the Open Archives Initiative (OAI) Protocol for Metadata Harvesting. It illustrates the use of this protocol in three projects: OAD, NDLTD, and CITIDEL. The OAI protocol extends traditional digital libraries services in a user-centered fashion. This is exemplified in the ESSEX search engine, which also enables multischeming browsing and quality-oriented filtering.

Using query probing to identify query language features on the Web by Bergholz and Chidlovskii addresses the problem of discovering the lexical processing and query language characteristics of an uncooperative search engine. Their research shows that a relatively small number of probe queries and simple classification algorithms are sufficient to discover a range of search engine characteristics, including stopword removal, stemming, phrase processing, and treatment of AND operators, much of the time. In evaluations with 19 search engines, features are discovered correctly about 75–80% of the time. Future research will be directed at improving classification accuracy, for example with better feature selection and improved classification algorithms.

The effect of database size distribution on resource selection algorithms by Si and Callan extends work reported in the main SIGIR conference. The conference paper reported on a new resource selection algorithm (ReDDE) that compensates for skewed distributions of database sizes more effectively than prior algorithms. The workshop paper develops new versions of the CORI and KL-divergence resource selection algorithms that better compensate for skewed distributions of database sizes. The three resource selection algorithms are compared on several testbeds with varying distributions of database sizes and relevant documents. The extended version of KL-divergence is about as effective as the new ReDDE algorithm. The extended CORI algorithm is better than the basic CORI algorithm, but is the least effective of the three.

Decision-theoretic resource selection for different data types in MIND by Nottelmann and Fuhr is also a companion to a paper that appeared in the main SIGIR conference. The conference paper reports on a decision-theoretic framework for text resource selection based on characteristics such as relevance, access time, and access costs. The workshop paper extends the approach to other data types, such as person name, year, and image, and exact-match and approximate-match retrieval methods. Two of the three methods presented for estimating the retrieval quality of a digital library can be applied to text and non-text data types. In spite of its generality, so far this approach to federated search of multimedia digital libraries has only been evaluated using text resources due to a lack of large, widely available multimedia resources.

Distributed Web search as a stochastic game by Khoussainov and Kushmerick addresses the problem of maximizing the performance ("profits") of a search service in an environment containing competing search services. Search engines are assumed to compete by deciding which markets to serve with their finite resources, and consumers are assumed to flock to the search engines that best meet their needs. This process can be modelled as a stochastic game in which parties have only partial information, there is a limited range of actions, and actions take time to have effects. Evaluations were done using data derived from 100 days of Web proxy logs from a large ISP; 47 search engines were involved. Experimental results indicate the effectiveness of the general approach, but also demonstrate artifacts due to assumptions. Future research will be on improving the models, and reducing the simplifying assumptions.

Collection fusion for distributed image retrieval by Berretti, Del Bimbo, and Pala describes a model-based approach to image data fusion (i.e., merging results from different image libraries). During an offline model learning stage training data is acquired by a form of query-based sampling in which queries are submitted to an image library, images are retrieved (with their library-specific scores), and normalized, library-independent scores are computed with a fusion search engine. When sampling is complete, images from each library are clustered into groups, and pairs of library-specific and normalized scores are combined to learn group-specific linear models. During interactive retrieval an image's score is normalized by finding the most similar cluster and using the model parameters associated with that cluster. The method is very fast and worked well in experimental evaluations.

New methods of results merging for distributed information retrieval by Wu, Crestani, and Gibb addresses the problem of merging results exploiting overlaps in different retrieval sets in order to achieve better performance. The new results-merging algorithms proposed take advantage of the use of duplicate documents in two ways: in one case they correlate the scores from different results, in the other they regard duplicates as increasing evidence of being relevant to the given query. An extensive experimentation suggests that these methods are effective.

Recent results on fusion of effective retrieval strategies in the same information retrieval system by Beitzel, Jensen, Chowdhury, Grossman, Goharian, and Frieder takes a new look at metasearch by studying it within a single retrieval system. Metasearch is known to improve retrieval results, but prior research often focused on fusion from different retrieval systems, which conflates effects due to different representations and retrieval models. In this study the representation is held constant. The results are unexpected. The number of documents that appear in multiple retrieval lists ("overlap documents") is considered a good clue to the effectiveness of data fusion; for example, the well-known CombMNZ method exploits this feature. However, it is a poor predictor in this setting, rewarding common "near miss" documents and penalizing "maverick" relevant documents found using only a single method. This paper encourages a more careful examination of representation vs. retrieval model effects in future metasearch research.

The MIND architecture for heterogeneous multimedia federated digital libraries by Nottelmann and Fuhr presents an architecture for distributed information retrieval. It consists of five types of components: graphical user interfaces, data fusion components, a dispatcher, proxies, and digital libraries. Proxies provide "wrapper" functionality for each digital library, providing common schemas and APIs for heterogeneous, multimedia, and possibly uncooperative digital libraries. Proxies also provide local resource selection using a cost-based, probabilistic framework, so retrieval scores are normalized across different media and digital libraries. The architecture provides varying levels of distribution, depending upon user needs. Communication among architecture components is performed using the SOAP protocol. An implementation is available.

Apoidea: a decentralized peer-to-peer architecture for crawling the World Wide Web by Singh, Srivatsa, Liu, and Miller describes a new spider architecture based on dynamic hash tables. Each node is responsible for a portion of the address (URL) space; each domain is covered by a single node, which keeps communication among nodes down to a manageable level. Exact duplicate detection is handled in a similar manner, by converting Web pages to hash values and making each peer responsible for a portion of the hash address space. The distributed approach makes it easy to distribute crawling geographically, possibly reducing communications costs. Initial experiments show very nearly linear scale-up as the number of nodes is increased.

Towards virtual knowledge communities in peer-to-peer networks by Gnasa, Alda, Grigull, and Cremers describes a peer-to-peer architecture consisting of personal digital libraries ("personal search memory" or PeerSy) and an architecture that lets them organize into virtual knowledge communities (VKCs). Virtual knowledge communities occur by clustering nodes based on each node's frequently asked and seldom asked queries and bookmarked documents (considered relevant). New queries are sent to one's personal digital library (PeerSy), one's virtual knowledge community, and Google. The expectation is that documents found within a person's personal digital library and virtual knowledge community will be better matches for an information need, possibly reflecting some combination of past searching and browsing behaviour. The work is at the initial prototype stage.

The personalized, collaborative digital library environment CYCLADES and its collections management by Candela and Straccia describes the CYCLADES system. In this system a digital library is not just an information resource where users submit queries to satisfy information needs, it is also a personalized collaborative working and meeting space. In this space users sharing common interests may organize the information space according to their own, subjective view. They may also build communities, become aware of each other, exchange information and knowledge with other users, and get recommendations based on preference patterns.

Considerations

Workshops on distributed information retrieval were held in conjunction with SIGIR 1996 and 1997 [1]. The response to the 2003 workshop indicates that many of the same issues remain important: for example, data gathering, resource selection, data fusion, and architectures. However, comparison with the earlier workshops also indicates that the topic has matured considerably. Assumptions about small numbers of cooperating, homogeneous resources running the same software are no longer pervasive. Resource selection algorithms are more accurate, more robust, and are beginning to really address multimedia data; fusion algorithms

[1] See `http://www.cs.cmu.edu/~callan/Workshops/nir96/` and `http://www.cs.cmu.edu/~callan/Workshops/nir97/`.

are much less ad hoc and much more effective; peer-to-peer architectures have emerged; and software architectures have become more detailed and realistic.

During the general discussion there was considerable debate about the state of resource selection research. Resource selection has been the driving topic in this research area for the last decade, and there has been steady improvement, but the upper bound remains unknown. Precision-oriented methods dominated past research, with much success, but high recall and high diversity are neglected topics that are particularly important in some domains, for example to better represent the range of information available.

Participants felt that data fusion research needs to continue its transition to stronger theoretical models. The field does not yet understand how differing levels of overlap among resources affect fusion algorithms; the research community is split into "much overlap" (e.g., metasearch) or "little overlap" (e.g., distributed IR), but the real world is more complex. The field also needs to learn to model the interaction between resource selection and data fusion. Improvements in resource selection may have a large effect or none depending on the data fusion algorithm, but today the interaction is unpredictable.

The topic that generated the most discussion was, of course, evaluation. There was broad agreement that there is too much focus on testbeds based on TREC data. Participants felt that it is especially necessary to model the size and relevance distributions of real sites, and that it might be possible to get such information from industry. There was recognition that different tasks and environments will have different characteristics, and that the research community needs to devote more effort to understanding what they are. A major obstacle for many researchers is that distributed IR is still rare in the "real world," so it is difficult to find "real" data, users, and failures. The clearest example of distributed IR in use today is in peer-to-peer networks such as KaZaA.

Relevance-based ranking (RBR) is a convenient and clear metric, but participants felt that the field will need to transition to a utility-based metric, possibly something like Norbert Fuhr's decision theoretic framework, that encompasses a wider range of user criteria. Such a transition will require a much better understanding of user information needs in distributed environments: for example, the importance of relevance vs. communication time, search vs. browsing, and relevance vs. diversity.

One could summarize the discussion of evaluation as a strong worry that researchers are stuck searching under the same old lampposts due to a lack of realistic data and user information needs. Participants expressed support for a TREC track or INEX-style project to focus attention on creating new datasets, task models, and evaluation metrics. The TREC-4 and TREC-5 Database Merging tracks were conducted before a distributed IR research community had developed, and hence they attracted little participation. Today, with active interest in distributed IR and federated search from a variety of research communities, a similar effort would have a much better chance of success.

Acknowledgements

We thank the organizers of SIGIR 2003 for their support of the workshop. We also thank the members of the Program Committee (Donatella Castelli, Jim French, Norbert Fuhr, Luis Gravano, Umberto Straccia) for their efforts on behalf of the workshop. The workshop was sponsored in part by the MIND Project (EU research contract IST-2000-26061), the University of Toronto, and the Knowledge Media Design Institute.

November 2003

Jamie Callan
Fabio Crestani
Mark Sanderson

Organizing Committee
SIGIR 2003 Workshop on
Distributed Information Retrieval

Table of Contents

Harvesting: Broadening the Field of Distributed Information Retrieval

Edward A. Fox[1], Marcos A. Gonçalves[1], Ming Luo[1], Yuxin Chen[1], Aaron Krowne[1], Baoping Zhang[1], Kate McDevitt[1], Manuel Pérez-Quiñones[1], Ryan Richardson[1], and Lillian N. Cassel[2]

[1]Digital Library Research Laboratory, Virginia Tech, Blacksburg, VA 24061 USA
{fox, mgoncalv, lming, yuchen, akrowne, bzhang, kmcdevit, perez, ryanr}@vt.edu

[2]Dept. of Computing Sciences, Villanova University, Villanova, PA 19085-1699 USA
lillian.cassel@villanova.edu

Abstract. This chapter argues that in addition to federated search and gathering (as by Web crawlers), harvesting is an important approach to address the needs for distributed IR. We highlight the use of the Open Archives Initiative Protocol for Metadata Harvesting, illustrating its use in three projects: OAD, NDLTD, and CITIDEL. We explain how traditional services can be extended in a user-centered fashion, providing details of our new: ESSEX search engine, multischeming browsing, and quality-oriented filtering (using rules and SVMs). We conclude with an overview of work in progress on logging and component architectures, as well as a summary of our findings.

1 Introduction

Distributed IR systems (DIRS) such as digital libraries (DLs) [1] involve rich collections of digital objects and community-oriented specialized services such as searching, browsing, and recommending. To ensure such comprehensive support of the needs of their target user communities, many DIRS are built using ensembles of autonomous, possibly heterogeneous systems, distributed across the Internet. This approach is often necessary because of social, economic, political, or intellectual property right concerns that preclude a simple centralized solution. Thus, a technology-based "fix" is needed that provides all the advantages of a centralized solution; users must have a transparent, integrated view of the component collections and information services.

Resulting challenges faced by DIRS include: 1) interoperability among different systems/protocols; 2) resource discovery (e.g., selection of the best sites to be searched); 3) issues in data fusion (merging of results of a search into a unique ranked list); and 4) aspects of quality of data and services.

Specific challenges faced by distributed IR systems which complicate interoperability and transparent resource discovery include:

- Autonomy: members locally manage their own content and services, and optimize resources and implementations to attend the needs of their local communities, which are not necessarily the same as the needs of other members.
- Decentralization: members are not asked to report either collection updates or changes in their metadata or services to central coordinators.
- Minimal interoperability: even if a minimal level of interoperability is supported like the provision of unique URNs and metadata records for all stored works, all members may not completely support the same standards or protocols.
- Heterogeneity: there is potentially enormous diversity in terms of natural language, metadata, protocols, repository technologies, character coding, nature of the data (e.g., numbers, characters, tables, texts, hypertexts, multimedia), as well as user characteristics, preferences, and capabilities.
- Scalability and dynamism: DIRS may have a huge number of members, new members may be constantly added, and there may be a continuing flow of new collections and resources.
- Domain-specific issues: issues specific to a domain or discipline also may play some role. For example, due to the primary-source nature of collections of electronic theses and dissertations (ETDs) [2], the site selection process that is found in other systems (identifying a small number of candidate databases to search) is not important in this context. A query asking for new results in mathematics could retrieve information from almost every member university, and a search for an interesting dissertation cannot ignore any member site lest it miss important relevant documents.

In this chapter we discuss how these challenges can be met. In the next section we show how our recommended approach, harvesting, compares with other approaches (federated search or gathering). Sect. 3 offers three case studies of how harvesting works in widely differing contexts. Sect. 4 explains a number of methods to enhance services when content is harvested. Finally, Sect. 5 introduces work-in-progress while Sect. 6 gives conclusions.

2 Harvesting, Federated Search, or Gathering

There are basically three approaches for DIRS interoperability and transparent resource discovery, which differ in the amount of standardization or effort required:

1. Federated services: In this approach a group of organizations (i.e., a "federation") agrees that services will be built according to a number of approved specifications, normally selected from formal standards. The problem of establishing a federation is the effort required by each organization to implement and keep current with all the agreements. For example, many libraries have standardized on the Z39.50 protocol to meet the needs for record sharing and distributed searching [3, 4]. The high costs of belonging to the federation of libraries that have adopted Z39.50 have been absorbed over decades, and they are balanced by the cost savings from shared cataloging [5]. The protocol specifies rules to allow a client to connect, search, request information (about available collections, formats, and syntaxes), and browse indexes of the server. Most implementations emphasize searches that use bibliographic attributes to search

databases of MARC catalog records and present them to the client. However, since the protocol is flexible, large, and complex, many implementations have features that others lack, and catalogs are internally organized and presented in different ways – all factors that limit interoperability.

Dienst [6], another example, used by the Networked Computer Science Technical Reference Library, was built on top of several Internet and Web technical standards (e.g., HTTP, MIME) [7, 8]. The protocol divided digital library services into four basic categories: repositories, indexes, collections, and user interfaces. In the case of distributed searching, a query was broadcast to all Dienst servers and the results of queries were combined locally by the user interface service. But, as the number of servers grew, scalability problems arose regarding server availability and network latency. Besides the fact that performance was determined by the "weakest link", failure to receive an answer from any server could mean loss of important information. Further, supporting the software implementation proved costly. NCSTRL has recently changed to a harvesting mode [9].

2. Harvesting: The difficulty of creating large federations provides motivation to create looser groupings of DIRS. The underlying concept is that the participants make some small efforts to enable some basic shared services, rather than adopting a complete set of agreements. When compared to federated services, the advantages of harvesting include performance and scalability since services can be locally built and optimized (e.g, local indexing of a union collection of the content of all federation members) and flexibility, for example, collections can be aggregated in many different ways (e.g., by genre – see Sect. 3.2, or discipline – see Sect. 3.1 and 3.3).

   Harvest^{TM} was an early system developed in accord with the principle of harvesting [10]. It defined a protocol and specific data format (SOIF, now being replaced by XML) to allow flexible and interoperable harvesting. It focused on the creation of topic-specific collections to avoid vocabulary and scaling problems of generic indexes. The architecture of Harvest^{TM} involves gatherers and brokers. Gatherers collect and extract indexing information from servers. Brokers provide the indexing mechanism and the query interface. Different brokers can be created by selectively choosing content from other gatherers or other brokers.

   The best current example of harvesting is illustrated by the Open Archives Initiative (OAI) [11, 12, 13]. OAI explicitly divides a DIRS into data providers, those wishing to publicly expose their metadata, and service providers, responsible for building information services atop the harvested data. The initiative promotes the use of Dublin Core [14] as a standard metadata format, but community-oriented metadata standards also can be used. There is a simple standard metadata harvesting protocol, the Open Archives Protocol for Metadata Harvesting (OAI-PMH). This protocol, built on top of HTTP, defines six verbs that allow a harvester to request information about an archive, incrementally request identifiers or complete metadata records in specific formats, or investigate the internal organization of data providers and their supported metadata formats. In spite of its simplicity, OAI adoption may be problematic, since it involves small amounts of coding and building of middleware layers for local repositories that sometimes do not match very well with the OAI infrastructure (e.g., repositories based on the Z39.50 protocol). Some recent efforts aim to further lower this barrier [15, 16]. More problematic is the fact that some archives are not willing to take any action beyond Web posting to further open their content, making gathering, the next approach, the only available option.

Further, while harvesting solves some of the problems of federated services, it adds new problems into the DIRS arena. Since collections are locally and autonomously managed, issues of data quality and completeness arise (see Sect. 4.4). Also, semantic interoperability, for example, due to the use of different classification schemes, makes it very complex to build unified services, like browsing (see Sect. 4.3). Another important issue is de-duping, or removal of duplicates or near-similar records, due to the aggregative nature of open archives, where (union) archives can be built by harvesting other archives. But this problem also arises in other contexts, so we consider it out of scope regarding the current discussion.

3. Gathering: If organizations are not prepared to cooperate in any formal manner, a base level of interoperability is still possible through the automated gathering of openly accessible information. The best examples of gathering are Web search engines [17]. Because there is no extra cost to those providing data, gathering can provide services that embrace large numbers of servers, but the services usually are of poorer quality than can be achieved by partners who cooperate more fully. These problems are mainly due to the quality of the data that can be gathered, including aspects of lack of structure and absence of provenance information as well as broadness of services. Even more complex is the issue of the "hidden web", where the contents of many valuable web-accessible databases are only accessible through search interfaces and are hence invisible to traditional web "crawlers" [18, 19].

In this paper, we concentrate on harvesting issues, mainly related to the Open Archives Initiative [11]. We highlight key problems and how several projects have addressed them.

3 Semi-dynamic Distributed Collections

In this section we illustrate the application of harvesting by way of three rather different case studies.

3.1 OAD: Harvesting from among Homogeneous Institutions

Introduction to the Project: The OAD (Open Archives: Distributed services for physicists and graduate students) [20] project aims to improve distributed digital library services to better support scholarly communication. It mainly focuses on two communities: physicists and graduate students. OAD is a German-US cooperation project launched simultaneously at Virginia Tech (by our lab) and ISN Oldenburg, Germany. It is funded jointly by the National Science Foundation (NSF) and the Deutsche Forschungsgemeinschaft (DFG). This project builds upon the German efforts led by E.R. Hilf to serve the physics community with better online information services. There are a number of expected benefits. First, it may help the physics community build / improve a large number of archives. Second, this project may ease the development of tools and materials related to ETDs and physics, as well as various software routines for OAI. Third, the process and toolkits developed in the project may be used to tailor an archive to a particular community of users. Fourth, software will be developed regarding mappings between metadata sets. Finally, improved and new services for PhysNet will be provided to the physics community.

PhysNet: Coordinated by E.R. Hilf, PhysNet [21] is a noncommercial, distributed information service for the physics community. Its target is to provide a longtime stable and distributed service for physics with cooperative help from many national and international societies and physics organizations. Information used in PhysNet is retrieved from the web servers of the worldwide distributed physics institutions and departments of universities, and thus forms a large distributed database. The quality and relevance to physics of the offered information is assured by its creators themselves at their local institution's web server. The services provided by PhysNet include PhysDep, PhysDoc, and PhysJob as well as links to resources from journals, conferences, and educational sites.

PhysNet harvesting: Among the major data providers in the physics community are: arXiv, APS, and IoP. The metadata collection from arXiv is freely available to the users, but the other two organizations require agreements prior to harvesting their metadata collections. Since OAI-PMH harvesting uses the HTTP protocol, authorization and authentication are a challenge. Possible solutions may include using the HTTPS protocol to transport user/password data. Another challenge is that the arXiv collection does not have labeled records, which makes text categorization problematic. The metadata collections we harvested from APS and IoP are in formats called aps-basic and stk-header. For experimental studies at VT related to PhysNet, the OCLC [22] OAI harvester is used. It is a Java application providing an OAI-PMH harvester framework. This framework can be customized to perform arbitrary operations on harvested data by implementing required Java interfaces. Thus, a change was made to redirect the harvested data from the screen to a file dump. Additionally, for IoP harvesting, the java.net socket was replaced. In particular, an open source HTTP client toolkit was used, because the IoP STACK service requires a sign-on authentication before harvesting.

PhysDep (Physics Departments Worldwide) offers lists of links to the servers of worldwide distributed institutions and departments of universities related to physics, ordered by continent, country, and town. The users can search or surf a set of lists of links to find the needed information, but harvesting the departmental information from one or several institution servers is still a big challenge to the physics community. One feasible solution is to promote the application of OAI-PMH, and set up a local or regional OAI-PMH data provider for each institution server so that users can easily harvest information from the desired source. In order to let users get updated information from physics institutions and departments of universities, a web service can be deployed to register the new OAI data provider interface and keep track of the updated status of the information at these servers.

Although most information for physics over the internet is written in English, information in the format of other languages should not be ignored. Multi-lingual support to harvest physics information from over the internet also should be extended.

3.2 NDLTD: Harvesting from among Heterogeneous Institutions

NDLTD [23] is an international non-profit educational and charitable organization. The goal of NDLTD is to promote better graduate education through sharing, supported by a digital library of electronic theses and dissertations (ETDs) [2] from all around the world [24]. Currently there are over 185 NDLTD members, among them 161 member universities (including 6 consortia) and 24 other institutions.

ETD union harvesting: The general structure of the Union Archive [25, 26] is shown in Fig. 1.

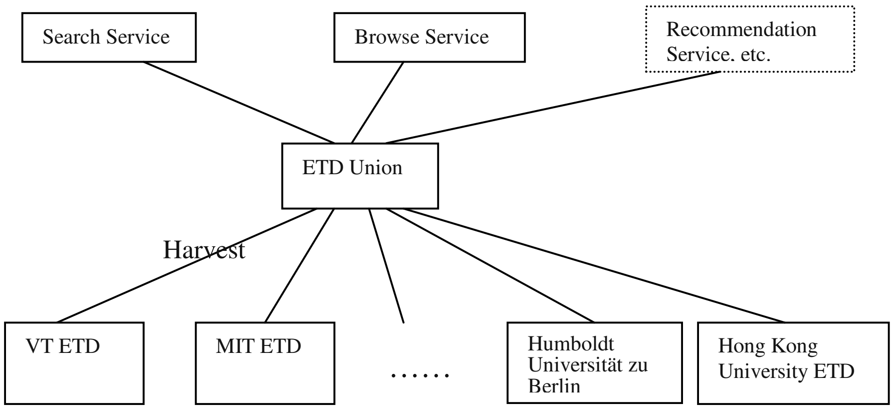


Fig. 1. Architecture of the ETD Union Archive

The ETD Union Archive originally used the union component from the Open Digital Library (ODL) [27] toolkit to harvest across all those collections. This union functionality then moved to OCLC (Online Computer Library Center in Dublin, Ohio, USA [22]). Since ETD data is distributed across individual universities located in many countries, the ETD Union Archive is built by harvesting from those collections. OAI-PMH increases the interoperability of heterogeneous ETD sites.

The individual ETD sites at different institutions are heterogeneous in terms of ETD software they are using, ETD formats, and the languages and character sets utilized. Some universities use VT-ETD software, some are exploring the applicability of DSpace [28], some employ BEPress tools [29], and some use other software. But as long as they comply with OAI-PMH, the union component can just treat them as a data provider. In this sense, OAI is bridging the gap of heterogeneous collections. NDLTD continues to grow rapidly. Governments and other institutions, in increasing numbers, realize the importance of sharing thesis and dissertation information.

A challenge to NDLTD harvesting is replicated records. Some sites have records in different formats (e.g., oai_dc, rfc1807, and etd_ms [30]). For data integrity, the best solution might be just keeping one form of data and generating different presentations from that form. But this increases the complexity of the upper layer software (which must support many users with rapid response). Right now, there is one copy kept in the database for each metadata format with the same OAI identifier.

Mirroring NDLTD worldwide: OCLC supports the union metadata collection, while VTLS Inc. and Virginia Tech run search/browse services using that data. H. Suleman's group in Cape Town University, South Africa is in charge of setting up a mirror site in South Africa with support arranged by A. Plathe of the Paris headquarters of UNESCO. H. Suleman's group also is redesigning the website for NDLTD.

China Academic Library and Information System (CALIS) [31], which is a public service provider supported by the Chinese Department of Education, representing a consortium of Chinese universities, is setting up another mirror in Beijing, China. The mirror sites are going to mirror both the NDLTD website and the ETD Union Catalog (plus searching and browsing services). In the future, there might be several regional centers (e.g., for North America, Asia, Africa, etc.) for NDLTD. Those centers could harvest from individual universities and then serve as a data provider to other centers. Thus harvesting can be distributed, and mirroring also can operate at those centers.

3.3 CITIDEL: Harvesting Prioritized by Size

CITIDEL (Computing and Information Technology Interactive Digital Educational Library [32]) is a project in the Collection Track of NSF's National Science Digital Library (NSDL) [33] program. Led by Virginia Tech, partners include the College of New Jersey, Hofstra, Penn State, and Villanova; support also is given by organizations such as ACM, IEEE-CS, NEC, and SUN.

CITIDEL aims to provide "one-stop-shopping" for teachers and learners in the computing and information technology fields. Where possible, it employs OAI-PMH to harvest information from a variety of sources. Since this protocol is not yet widely deployed, some effort typically is required by the project team to work with each new prospective source. At issue, then, is the order in which new sources should be added, since the project team is resource constrained and so must prioritize its efforts. Setting priorities is complicated by various practical considerations, and by the nature of the size distribution of content sources, and by variations in their quality. Balance is hard to achieve when adding new content, since there are few large sources, a moderate number of medium-size sources, and a large number of small-size sources.

Beyond the initial set of sources which project co-PIs work on and so can supply metadata for (e.g., CS Virtual History Museum [34], Computer Science Teaching Center [35], NCSTRL [7], PlanetMath [36], and NDLTD [23]), others have been added largely in order of priority by size, so as to maximize the benefit with regard to the relatively fixed amount of work required to arrange for and add a new collection. Through efforts by the CITIDEL team and by students working on projects, additional collections have been added from DBLP [37] and ACM CHI tutorials, while a number of others are under way, including IEEE-CS [38], eBizSearch [39], CiteSeer [40]. Many other efforts are proceeding with regard to CITIDEL, some touched upon in the following sections.

4 Expanded Services

One key advantage of harvesting is that it makes possible improved and expanded services, relative to those offered through other approaches. The following subsections illustrate the possibilities, beginning with approach, and continuing with examples.

4.1 User-Centered

Digital libraries can contain a wealth of information but are useless if this information cannot be retrieved and used. Two possible users of the data in digital libraries are computer systems and humans. Computer systems that can use the information include agents, special services, repositories, archives, and even other digital libraries. There are many standards in place to ensure that information retrieval between systems goes smoothly and consistently. These systems may use portions of the data, or have their own mechanisms for organizing and using. Likewise, human users of digital libraries and similar systems may all have different ways of thinking about and using information contained therein.

Developers should take into account what sort of users the system hopes to receive, when they first begin to design the interface, as well as later as the site expands. Keeping users' intentions and expectations in mind in the design stage is crucial and is especially important when the information is of a specialized nature such as with CITIDEL. It is important to decide how much information a user needs to see at a given moment, and how much information a user might be expected to supply when submitting an entry or collection to the digital library. Another important aspect is creating a logical division between the interface and the data. Maintaining a separation between the stored information and the part of system the user interacts with also makes it much easier to add future tools and features to the system that will help different users find and use resources. CITIDEL, for example, is a specialized collection of educational computing resources and yet targets a very wide user base. Professors and instructors of all levels may wish to use information in classes or when creating new lesson plans. Students from grade school to graduate school might seek resources to be used for projects or personal understanding. Researchers are able to find different sorts of information that might be useful to them whereas professionals may use the same resources for different reasons. Hence, when designing and developing for access and use, it is crucial to consider the age, professions, education, computer familiarity, digital library familiarity, and interests of users.

Not all users may want to retrieve information in the same way. CITIDEL is a way to get to many different resources through one system, but finding a specific piece of information within might take multiple, advanced searches or refining and filtering techniques. Other users might have a less specific idea in mind of what they would like, and instead wish to use a browsing scheme they are familiar with to find the information (see Sect. 4.3).

Likewise, not all users may wish to use the resources in a digital library in the same way. For example, an interactive applet could be used by a student at home to learn more, or might be used in class by a professor as a demonstration. One researcher might prefer keeping a collection of papers on a topic in his online CITIDEL binder whereas a professor might find a group of resources and use the VIADUCT (Virginia Instructional Architect for Digital Undergraduate Computing Teaching) tool in CITIDEL to create a lesson plan using those resources which he then can distribute to others. Many features and tools in CITIDEL have been added for the user to not only find resources of interest but also to use them. Users may create active lists (ordered and annotated lists of resources) or easily discuss a resource's content and possible uses in its own associated discussion thread.

In order to create a digital library or other system which provides a useful service, it is imperative to construct both resource retrieval and resource usage around what the different sorts of users of the system want and need. Through both user-centered design and frequent interactive usability evaluations, we have worked, as can be seen in the next subsections, to create a system that is usable by the target audience.

4.2 Searching

CITIDEL leverages the harvesting approach to allow searching of its distributed resources. All of the metadata for CITIDEL records is stored locally in a union catalog, periodically updated from the original sources. Using the ESSEX search engine [41] an entirely in-memory index is maintained, allowing for fast searches of the metadata. This index is kept small by a default postings list entry size of four bytes (in our deployment, requiring about 500MB for a half-million records). In essence, the ESSEX component "localizes" the distributed search problem by being efficient enough to sit atop a union catalog on a system with only modest server-class resources.

The ESSEX search engine was developed for the CITIDEL project, with an eye towards serving as the core of an ODL [27, 42, 43] search component. Such a component would be higher-performance and more scalable than the existing ODL IRDB search component, and also eliminate the need for a DBMS to provide for the low-level storage of index information. ESSEX was designed modularly to plug into the existing ODL IRDB code base without changing or affecting the interface, thus making an ESSEX-based ODL component a drop-in replacement.

To achieve this modular architecture, ESSEX was designed to run as a daemon server, answering requests from client programs via a socket interface (either TCP or UNIX-domain). In order to drop ESSEX into the existing IRDB Perl code base, a Perl client library was first written. However, client libraries could be written for any language, allowing a digital library implemented in languages other than Perl to use ESSEX directly as a search engine core. We plan on providing more pre-written client libraries for ESSEX in the future, with Python likely early on.

ESSEX is a vector-space search engine which provides a number of features that make it optimal for use in digital libraries. Underlying many of these features is the support for metadata fields. In ESSEX, all information is indexed in “chunks” associated with field names. These chunks may correspond to XML elements in metadata. ESSEX is completely adaptive, not requiring a declarative specification of the DL’s metadata. The digital library client must simply determine for itself what the searchable subset of the metadata is, and send the appropriate portions to ESSEX to be indexed.

Built on top of this adaptive metadata field-aware basis are a number of query language provisions. Maintaining the standard set supported by IRDB, ESSEX allows narrowing of search terms to metadata fields, as in “title:distributed”. Here, the field name precedes the colon, and the atom is interpreted as “only occurrences of ‘distributed’ in the ‘title’ field”. This can be combined with force/forbid operators (‘+’, ‘-‘), as in “+title:distributed”, which would tell ESSEX to retrieve only documents which strictly contain occurrences of “distributed” in the “title” field.

In addition to narrowing and filtering, fields allow ESSEX to perform intelligent weighting. Giving preferential weightings to certain portions of metadata (such as title) may result in more effective retrieval. To provide for this, ESSEX combines standard query relevance values with a field weight factor, rewarding results which are “rich” in highly-weighted fields. Default weightings are expected to be supplied by the digital library deployer, but they also can be customized within the query by adding “field=weight” style atoms. The values of ‘weight’ act relatively, with the default weight being one. When all field weights are one, the ranks returned by ESSEX are identical to standard cosine-normalized vector space ranks.

Using ESSEX’s index disk snapshot support, re-indexing of the collection metadata need not be done after system crashes or shutdowns. This allows for the run-time speed benefits of the current in-memory-only architecture, but without a large start-up performance hit for large collections.

4.3 Browsing through Multischeming

It is highly desirable to provide today, as a basic digital library service, a “directory” style browsing interface. These interfaces are driven by classification schemes (or subject hierarchies [44]), with famous examples being Yahoo [45] and Open Directory [46]. In a digital library, such a browsing interface allows the patron to quickly select categorically-related groups of resources in a progressive fashion, moving from general to particular. These resources are classified (or categorized), meaning metadata is present which associates the records with categories in the hierarchy.

In a distributed setting, complications appear. We can hope, but not guarantee, that all of the federated digital libraries use the same classification scheme. Coming from different origins at different times, and serving different regions and sub-communities, multiple classification schemes for the same domain can appear. A digital library which uses a single classification scheme to place resources in a browse directory may find itself unable to place in the same directory resources which it has harvested, due to classification under a different scheme.

There are a number of unsatisfactory ways to accommodate this problem. We could standardize on a scheme within the entire federation. This has the benefit of uniformity and simplicity, but could be a monumental political and technical challenge. The more alternate schemes there are, the more patrons there will be who will be displeased at having to abandon what they are familiar with. Alternatively, we might support a superset of all schemes in a parallel manner, providing a separate browse directory for each scheme. This, however, only solves the problem of resources being "hidden"; it does not solve the problem of users having to learn many schemes. It also introduces a new complexity in the search for a particular item – that of finding which of an arbitrary set of schemes a resource is classified under. Another solution is to combine the parallel-directories method with adding categorizations for each resource for all schemes. This would ensure all resources appear under each parallel directory and not require users to learn any new schemes, but producing the classifications is likely to be difficult, and perhaps intractable.

In CITIDEL, we have chosen to solve the problem through a scheme-agnostic multi-scheming system. It resembles the last solution mentioned – that of parallel directories and multiple classifications – but avoids the work of producing any additional classifications for the resources themselves. Instead, all that is required is a set of mappings between classification schemes, consisting of "edges" between categories across schemes which are semantically equivalent or supersets of each other.

An example category mapping from the computing field might declare that "CC2001.IS" (Computing Curricula, Intelligent Systems) contains "CCS1998.I.5" (Computing Classification Scheme, Pattern Recognition).

From a small number of these mappings, a scheme index can be built. This index allows the digital library to translate the request "return all resources in the category X" to "return all resources in the category X or any categories which it contains". This index is transitive, so the graph of schemes which are mapped to each other need only be connected. That is, we need map each scheme to only one other.

With this system, each resource, regardless of which scheme it is classified under, is visible when browsing via each of the different schemes through the directory interface. We have implemented this system in CITIDEL, and produced sufficient mappings for the data we have so far. To evaluate the effectiveness of our multischeming approach, we measured "virtual classifications". These are "simulated" classifications, which would need to be added to each record's metadata manually to achieve the same end effect as the multischeming system. Results are shown in Table 1.

Table 1. Statistics Regarding Effectiveness of Multischeming in CITIDEL

Classified resources	89,150
Classifications	241,723
Average classifications/object	2.7
Virtual classifications	1,968,488
Average virtual classifications/object	22.1

Table 2 presents some scheme statistics for CITIDEL. Combining the information from these two tables, we can see that multischeming has allowed us to leverage 244 category mappings to balloon 241,723 classifications into 1,968,488 virtual classifications, multiplying 2.7 average classifications per resource to 22.1, an improvement by a factor of almost eight times. When one considers that producing each category mapping is approximately as labor-intensive as producing a new classification for a single resource, it is clear that the overall savings are enormous.

Table 2. Multischeming Summary Statistics for CITIDEL

Schemes	4
Total categories	6,166
Total category mappings	244

Thus, multischeming presents a useful means to solve the problem of browsing of distributed resources in a union catalog setting, in spite of the existence of potentially multiple conflicting classification schemes. The value of the end-user's existing experience is preserved, without requiring the DL maintainer to perform a Herculean amount of work in metadata generation. For more details on multischeming and its implementation in CITIDEL, see [47].

4.4 Filtering, Quality of Data

Motivation: One goal of CITIDEL is to maximize the number of computing-related resources available to computer science scholars and practitioners. In order to achieve this goal of maximum coverage of computing related information, a sub-collection of computing related electronic theses and dissertations (ETDs) was filtered out from the Networked Digital Library of Theses and Dissertations (NDLTD) [23, 48] OAI Union Catalog [25, 26] and added to CITIDEL.

Metadata Quality Analysis: One way to filter out computing entries from the whole NDLTD collection would be to look at the contributor fields of each record (in the case of ETDs where this entry corresponds to mentors and/or committee members). For example, if most of the contributors are computer scientists, one could expect that this record is computing related. Another way is to look at the subject fields.

However, after analyzing the metadata quality of the Union Catalog [25], we noticed that only about 65% of the entries have contributor fields while about 92% have subject fields. We also noticed the inconsistency of using contributor and subject fields across universities. Specifically, some university's records are missing contributor fields, thus making impossible the filtering task using only contributor.

Though the subject fields are most useful overall in filtering, their sole use is prevented by a number of factors. The first is that many records at several sites are missing subject fields. The second factor is the inconsistency in use of subjects at several universities as an indicator of topical field. For example, some records may use subject field to represent department name, while other records use subject field to indicate key words; we see that different universities use subject fields in different ways. Thus, in such an open-ended research collection, it's not possible to enumerate a list of words that will include all possible computing related entries for different universities.

To get better filtering results, we need to combine several sources of evidence. If an instance lacks both the contributor field and subject field, we only can rely on the content based classification as evidence of the record being computing related (i.e., text from title and/or abstract). Accordingly our filtering process was composed of a combination of three major steps:

1. Content-based classification – classify the data set into two sets: computing related and non-computing related – using title plus subject plus abstract;
2. Filtering based on contributor field(s) in the metadata – enhance the results achieved through content-based classification;
3. Filtering based on subject field(s) in the metadata – enhance results when records of the second step are missing contributor fields or to correct possible records mistakenly filtered out.

Content-based Classification: We chose to use the WEKA SMO [49] classifier. The SMO classifier implements the sequential minimal optimization algorithm. It is a fast method to train Support Vector Machines (SVMs) which has reportedly been the best classifier so far for text classification experiments [50, 51, 52, 53].

Figure 2 shows the composition of the training and test sets for the experiments. Normal arrows represent how we selected the training collection and the test collection. Dotted arrows represent the input to the training and classification process. Dashed lines represent the output of the training process.

We used records extracted from the ACM Digital Library [54] (included in CITIDEL) to provide positive training examples to our classifier instead of some other computing related records because the previously identified computing sub-collection in NDLTD only has 138 entries. We felt this sub-collection was too small to serve as the computing category of the training set.

Contributor Filtering: To remove false computing entries from the classification result set, we decided to look further at the contributor fields for those classified as computing entries. For an ETD, the contributors are normally members of the committee. A simple heuristic was defined wherein if more than half of the contributors are computer scientists, this thesis or dissertation is assumed to be computing related. We used a set of author names extracted from records of the ACM DL as our computer scientists list.

Subject Filtering: From the analysis of NDLTD records, it was noted that contributors were not used uniformly and consistently across institutions (recall Sect. 4.4.2). Therefore many entries may not be filtered using only the contributor filter. To deal with this situation, we applied a second filter on those entries missing contributor fields. We analyzed the metadata of those entries and found that 71 of 88 entries have subject fields. We noted that some non-computing words are subjects. For example, words like ‘physics’, ‘animal science’, ‘Landscape Architecture’, etc. appear. So we could use them to remove entries whose subject field had such kinds of words. We also noticed that when the contributor filter was applied on the classified computing collection, a few positive entries were filtered out. So we also applied the subject filter in another way on these entries. That is, we check if these entries’ subject field contains words like ‘Computer Science’ and if they do, we move them back into the computing collection.

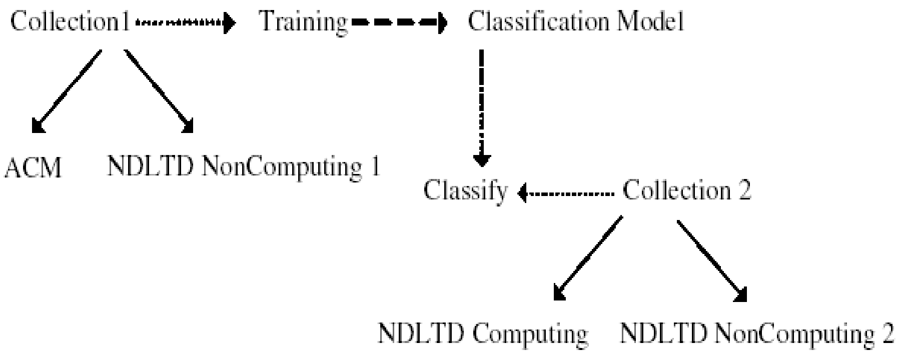


Fig. 2. *Training set* and *Test set* composition

Filtering Process Results: All the experimental results are summarized in the recall and precision measures in Fig. 3. Precision is defined as the proportion of correctly classified records in the set of all records assigned to the target class. Recall is defined as the proportion of correctly classified records out of all the records having the target class.

Content based classification yields good recall but low precision. After applying a subject filter to the content based classification, we improved the precision without loosing recall. When we applied a contributor filter to the content based classification result, we also improved precision but at the same time filtered out some computing entries, so recall was reduced. On the other hand, after applying the subject filter, we could remove more false computing entries and at the same time bring back some real computing ones. So considering both recall and precision, we achieved the best results by combining these three types of evidence.

Our experiments show that the quality of the filtering service was affected by the quality of the metadata. Several sources of evidence were combined to overcome the problem of inconsistency of the instances' metadata and to yield better results. To address the problems of absence, inconsistency, and ambiguity of contributor and subject information in NDLTD, content-based classification was studied. This procedure gives good recall but low precision. The contributor filter tried to look at the contributor field(s) and to filter out non-computing related instances. And to address the problem of lacking contributor field(s) for some instances, the subject filter tried to look at the subject information to filter out non-computing related instances. The final combination of the filtering mechanisms provided satisfactory precision and recall.

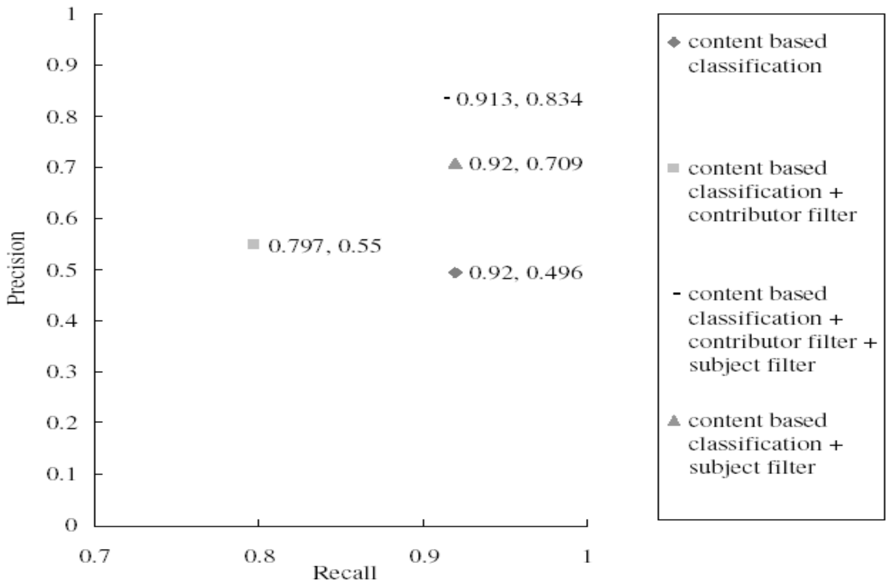


Fig. 3. Experimental results in *Recall-Precision* format

5 Work-in-Progress

The following two subsections discuss work underway that stretches beyond current harvesting efforts. The first is necessary for evaluation and improvement of DIRS, while the second makes it more feasible to build complex DIRS, through the application of modern software engineering methods.

5.1 Distributed Logs

The benefits of a distributed library depend on close coordination of the operations related to the overall goals. Earlier sections have described philosophies and techniques for harvesting information and for providing expanded services in the context of a distributed library. Among the needed resources to support a coordinated effort are logs of the activities of the component parts of the whole.

Logging provides pictures of the status of the individual systems both at distinct points in time and as a longitudinal record over a period of time. Some type of logging is available in any system designed for performance monitoring. The complication in the case of a distributed library is that the various components may have incompatible, non-interoperable logging systems, thus making it difficult or impossible to get a comprehensive picture of the performance of the aggregate system.

We have developed an XML-based log standard for use in digital libraries [55, 56]. The log standard includes over 50 standard event tags such as browse and login. The standard is extensible, allowing additional features to be added if they are required in a particular environment. The elements of the standard can be embedded within each other, leading to customizable and very readable records of user and system events.

We also consider the analysis of the logs generated when the XML log standard is installed. Clickstream analysis tracks user activity and provides indications of user success or difficulty. User privacy is respected by substituting a numeric identifier for traceable information such as the login name or IP address. Analysis shows where user activity focuses, time spent in browsing, number of resources touched, length of stay, frequency of return (if a registered user), choices made among resource options, etc. Resource hit counters document which resources are most in demand. Analysis also shows system performance, including crashes or failures to respond to a user request.

The significance of the log standard in a distributed library is twofold: First, the log standard provides a much richer collection of information about the performance of the system with regard to user activity than do traditional server logs; second, if the same log information is gathered by every element of the distributed library, only then can a meaningful evaluation of the overall performance of the entire library be considered. Work proceeds on development of easily portable modules that allow the log standard and its analysis to be inserted into an existing digital library as well as to be included in the component pool from which a library is constructed. As the standard log tools spread, meaningful evaluation will be facilitated, both of individual libraries and of libraries built of interoperable partner units.

5.2 Distributed IR Services Based on Component Architectures

So far, this chapter has considered issues related to distribution of content where most of the services were created in a centralized fashion. A recent research trend has been to distribute the IR *services* themselves along a number of cooperating units of distributed components. This trend has been fueled by the necessity to achieve a number of desired properties in DIRS including:

– Expandability: Distributed IR services should be able to easily incorporate new functionalities due to development of new technologies or due to demand for new usages of the system by specific communities or application areas. It should be possible to expand the service functionality just by adapting existing components or adding new ones without rebuilding the whole system.
– Scalability: Distributed IR services should be able to support the handling and processing of new content and capable of satisfying the needs of new user communities. Replication, coordination, mutual configuration and workload distribution are issues that should be taken into consideration.
– Quality of Service: QoS is an important issue in distributed systems. Dimensions of quality of service include performance, in terms of latency and throughput, reliability, cost, security, etc. DIRS should be able to sustain stable and efficient QoS by dynamic de-selection, re-selection, and grounding to different components when a previously selected one ceases to provide the desired quality.

Digital libraries have pioneered some of the first architectures of these future distributed IR systems. ODL [27] and DL-in-a-box [57] have a number of DL distributed components that implement services such as searching, browsing, recommending, etc. Those components communicate through an extension of the OAI-PMH and components can be composed together to implement new functionalities. The OpenDLib project [58, 59] follows the same philosophy but goes one step further. All OpenDLib service components are completely configurable, can be centralized, distributed or replicated, and there is a generic manager service capable of supporting "on-the-fly" dynamic service expansion. Finally, Web Services [60, 61] and the Semantic Web [62] are new technologies that have the potential to enable the implementation of this next generation of DIRS.

6 Conclusions

This chapter has highlighted our research on Distributed IR Systems (DIRS) in general, but in particular has focused on showing how the field can be broadened when a harvesting approach is employed. We have demonstrated how harvesting can work through three case studies: OAD, NDLTD, and CITIDEL. We have shown how expanded user-centered services can be provided, with the new ESSEX search engine, browsing through multischeming, and quality-oriented filtering (using rules and SVMs). Regarding work in progress, we have summarized our efforts regarding logging and component architectures. In conclusion, we argue that harvesting is a viable approach to DIRS, and urge other groups to consider adopting, adapting, and extending some of our tools/results.

Acknowledgements. Acknowledgment is given to support from AOL (for fellowship funding for M. Gonçalves, also supported by CAPES, process 1702-98), SUN (for equipment supporting CITIDEL), DFG (for support of our German collaborators on the OAD project), and to NSF funding of DUE projects 0121679, 0121741, 0136690 and IIS projects 9986089, 0002935, 0080748, 0086227. Thanks also go to members of the Digital Library Research Laboratory and all others involved in the various projects touched upon in this chapter.

References

1. Fox, E., Urs, S. Digital Libraries. Annual Review of Information Science and Technology, ed. Blaise Cronin, 36(12):503–589 (2002)
2. Fox, E., Feizbadi, F., Moxley, J., Weisser, C., eds. The ETD Sourcebook: Theses and Dissertations in the Electronic Age, New York: Marcel Dekker (2004, in press)
3. National Information Standards Organization. Z39.50: Information Retrieval (Z39.50): Application Service Definition and Protocol Specification. Bethesda, MD: NISO Press (1995)
4. Moen, W. E. Accessing Distributed Cultural Heritage Information. CACM 41(4):45–48 (April 1998)

5. Arms, W. Y. Digital Libraries. Cambridge, MA: MIT Press (2000)
6. Lagoze, C., Davis, J. R. Dienst – An Architecture for Distributed Document Libraries. CACM, 38(4):47 (April 1995)
7. NCSTRL. Networked Computer Science Technical Reference Library. Homepage. http://www.ncstrl.org (Available Nov. 3, 2003)
8. Lagoze, C., Fielding, D., Payette, S. Making global digital libraries work: collection services, connectivity regions, and collection views. Proceedings of the Third ACM Conference on Digital Libraries: 134–143 (1998)
9. Anan, H., Liu, X., Maly, K., Nelson, M., Zubair, M., French, J., Fox, E., Shivakumar, P. Preservation and transition of NCSTRL using an OAI-based architecture. JCDL 2002: 181–182
10. Bowman, C., Danzig, P., Hardy, D., Manber, U., Schwartz, M. The Harvest Information Discovery and Access System. Computer Networks and ISDN Systems 28(1&2):119–125 (1995)
11. OAI. Open Archives Initiative. Homepage. http://www.openarchives.org (Available Nov. 3, 2003)
12. Lagoze, C., van de Sompel, H. The Open Archives Initiative: building a low-barrier interoperability framework. JCDL 2001:54–62
13. Suleman, H., Fox, E. The Open Archives Initiative: Realizing Simple and Effective Digital Library Interoperability. In special issue on "Libraries and Electronic Resources: New Partnerships, New Practices, New Perspectives" of J. Library Automation, 35(1/2):125–145 (2002)
14. Dublin Core Metadata Initiative. Homepage. http://dublincore.org/ (Available Nov. 3, 2003)
15. Hochstenbach, H., Van de Sompel, H. The OAI-PMH Static Repository and Static Repository Gateway. JCDL 2003: 210–217
16. Calado, P., Gonçalves, M., Fox, E., Ribeiro-Neto, B., Laender, A., da Silva, A., Reis, D., Roberto, P., Vieira, M., Lage, J. The Web-DL Environment for Building Digital Libraries from the Web. JCDL 2003: 346–357
17. Wolf, J. L., Squillante, M. S., Yu, P. S., Sethuraman, J., Ozsen, L. Optimal crawling strategies for web search engines. WWW 2002: 136–147
18. Ipeirotis, P., Gravano, L., Sahami, M. Probe, Count, and Classify: Categorizing Hidden Web Databases. SIGMOD Conference 2001
19. Lage, J. P., Silva, A. S., Golgher, P. B., Laender, A. H. F. Collecting hidden web pages for data extraction. WIDM 2002: 69–75
20. OAD. Open Archives : Distributed services for physicists and graduate students. Homepage. http://www.dlib.vt.edu/OAD/ (Available Nov. 3, 2003)
21. PhysNet. The Worldwide Physics Departments and Documents Network. Homepage. http://www.phys.vt.edu/PhysNet/ (Available Nov. 3, 2003)
22. OCLC. Online Computer Library Center. Homepage. http://www.oclc.org (Available Nov. 3, 2003)
23. Fox, E.A. Networked Digital Library of Theses and Dissertations (NDLTD). Homepage. http://www.ndltd.org (Available Nov. 3, 2003)
24. Suleman, H., Atkins, A., Gonçalves, M.A., France, R.K., Fox, E.A., Virginia Tech; Chachra, V., Crowder, M., VTLS, Inc.; and Young, J., OCLC: Networked Digital Library of Theses and Dissertations: Bridging the Gaps for Global Access – Part 1: Mission and Progress. D-Lib Magazine, 7(9) (2001)
http://www.dlib.org/dlib/september01/suleman/09suleman-pt1.html (Available Nov. 3, 2003)
25. NDLTD Union Catalog Project. Electronic Thesis/Dissertation OAI Union Catalog Based at OCLC. Homepage.
http://rocky.dlib.vt.edu/~etdunion/cgi-bin/OCLCUnion/UI/ (Available Nov. 3, 2003)

26. Suleman, H., Luo, M. Electronic Thesis/Dissertation OAI Union Catalog. Homepage. http://purl.org/net/etdunion (Available Nov. 3, 2003)
27. ODL. Open Digital Libraries. Homepage. http://oai.dlib.vt.edu/odl/ (Available Nov. 3, 2003)
28. DSpace Federation. DSpace at MIT. Homepage. http://www.dspace.org/ (Available Nov. 3, 2003)
29. BEPres. The Berkeley Electronic Press. Homepage. http://www.bepress.com/ (Available Nov. 3, 2003)
30. ETDMS. ETD-MS: An Interoperability Metadata Standard for Electronic Theses and Dissertations. Homepage. http://www.bepress.com/ (Available Nov. 3, 2003)
31. CALIS. China Academic Library & Information System. Homepage. http://www.calis.edu.cn (Available Nov. 3, 2003)
32. CITIDEL. Homepage. http://www.citidel.org (Available Nov. 3, 2003)
33. NSDL. National Science Digital Library. Homepage. http://www.nsdl.org (Available Nov. 3, 2003)
34. On-line Virtual Computer History Museum. Homepage. http://virtualmuseum.dlib.vt.edu (Available Nov. 3, 2003)
35. CSTC. Computer Science Teaching Center. Homepage. http://www.cstc.org (Available Nov. 3, 2003)
36. Krowne, A. An Architecture for Collaborative Math and Science Digital Libraries. Masters thesis, Virginia Tech Dept. of Computer Science, Blacksburg, VA 24061 USA. http://scholar.lib.vt.edu/theses/available/etd-09022003-150851/ (Available Nov. 3, 2003)
37. Ley, M., ed. dblp.uni-trier.de Computer Science Bibliography. Homepage. http://www.informatik.uni-trier.de/~ley/db/ (Available Nov. 3, 2003)
38. IEEE-CS. IEEE Computer Society Digital Library. Homepage. http://www.computer.org/publications/dlib/ (Available Nov. 3, 2003)
39. eBizSearch. Homepage. http://gunther.smeal.psu.edu/ (Available Nov. 3, 2003)
40. NEC Research Institute CiteSeer : Scientific Literature Digital Library. Homepage. http://citeseer.org (Available Nov. 3, 2003)
41. Krowne, A. The ESSEX Search Engine. http://br.endernet.org/~akrowne/elaine/essex/ (Available Nov. 3, 2003)
42. Suleman, H. Open Digital Libraries, PhD Dissertation, Virginia Tech, 2002. http://scholar.lib.vt.edu/theses/available/etd-11222002-155624/ (Available Nov. 3, 2003)
43. Fox, E., Suleman, S., Luo, M.: Building Digital Libraries Made Easy: Toward Open Digital Libraries. ICADL 2002, Singapore
44. Dumais, S., Chen, H. Hierarchical classification of Web content. In Proceedings of the 23rd annual international ACM SIGIR conference on research and development in information retrieval (Athens, Greece), 256–263 (2000)
45. Yahoo! Homepage. http://www.yahoo.com (Available Nov. 3, 2003)
46. dmoz. Open Directory Project. Homepage. http://www.dmoz.org (Available Nov. 3, 2003)
47. Krowne, A., Fox, E. An Architecture for Multischeming in Digital Libraries, Virginia Tech Dept. of Computer Science Technical Report TR-03-25, Blacksburg, VA (2003) http://eprints.cs.vt.edu:8000/archive/00000692/ (Available Nov. 3, 2003)
48. Fox, E.A. Networked Digital Library of Theses and Dissertations. Nature Web Matters 12 (Aug 1999) http://helix.nature.com/webmatters/library/library.html (Available Nov. 3, 2003)
49. Platt, J. Fast Training of Support Vector Machines using Sequential Minimal Optimization. Advances in Kernel Methods – Support Vector Learning, B. Schölkopf, C. Burges, and A. Smola, eds., Cambridge, MA: MIT Press (1998)

50. Dumais, S. T., Platt, J., Heckerman, D., and Sahami, M. Inductive learning algorithms and representations for text categorization. In Proceedings of CIKM-98, 7th ACM International Conference on Information and Knowledge Management (Bethesda, MD), 148–155 (1998)
51. Joachims, T. Text categorization with support vector machines: learning with many relevant features. In Proceedings of ECML-98, 10th European Conference on Machine Learning (Chemnitz, GE), 137–142 (1998)
52. Joachims, T. A statistical learning model of text classification for support vector machines. In Proceedings of the 24th annual international ACM SIGIR conference on research and development in information retrieval (New Orleans, LA), 128–136 (2001)
53. Sebastiani, F. Machine learning in automated text categorization. ACM Computing Surveys (CSUR), 34(1):1–47 (2002)
54. ACM Digital Library. Homepage. http://www.acm.org/dl (Available Nov. 3, 2003)
55. Gonçalves, M. A., Luo, M., Shen, R., Ali, M. F., Fox, E. A. An XML Log Standard and Tool for Digital Library Logging Analysis. ECDL 2002: 129–143
56. Gonçalves, M. A., Panchanathan, G., Ravindranathan, U., Krowne, A., Fox, E. A., Jagodzinski, F., Cassel, L. N. The XML Log Standard for Digital Libraries: Analysis, Evolution, and Deployment. JCDL 2003: 312–314
57. DLbox Team. Digital Libraries in a Box. Homepage. http://dlbox.nudl.org (Available Nov. 3, 2003)
58. Castelli, D., Pagano, P. OpenDLib: A Digital Library Service System. ECDL 2002: 292-308
59. Castelli, D., Pagano, P. A System for Building Expandable Digital Libraries. JCDL 2003: 335–345
60. W3C. Web Services Architecture. Homepage. http://www.w3c.org/TR/ws-arch (Available Nov. 3, 2003)
61. Papazoglou, M. P., Georgakopoulos, D. Service-Oriented Computing, Special Section. CACM 46(10) (Oct. 2003)
62. Berners-Lee, T., Hendler, J., Lassila, O. The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. Scientific American, 279(5):35-43 (May 2001)
http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21 (Available Nov. 3, 2003)

Using Query Probing to Identify Query Language Features on the Web

André Bergholz and Boris Chidlovskii

Xerox Research Centre Europe (Grenoble)
6 chemin de Maupertuis
38240 Meylan, France
{Andre.Bergholz,Boris.Chidlovskii}@xrce.xerox.com

Abstract. We address the problem of automatic discovery of the query language features supported by a Web information resource. We propose a method that automatically probes the resource's search interface with a set of selected probe queries and analyzes the returned pages to recognize supported query language features. The automatic discovery assumes that the number of matches a server returns for a submitted query is available on the first result page. The method uses these match numbers to train a learner and generate classification rules that distinguish different semantics for specific, predefined model queries. Later these rules are used during automatic probing of new providers to reason about query features they support. We report experiments that demonstrate the suitability of our approach. Our approach has relatively low costs, because only a small set of resources has to be inspected manually to create a training set for the machine learning algorithm.

1 Introduction

Searching for relevant information is a primary activity on the Web. People search for information using general-purpose search engines, such as Google or AltaVista, which crawl and index billions of Web pages. However, there exists a fragment of the Web that is unavailable for central indexing. This so-called "hidden" part of the Web includes the content of local databases and document collections accessible though search interfaces offered by various small- and middle-sized Web sites, including company sites, university sites, media sites, etc. The size of the Hidden Web is estimated to be about 500 times bigger than that of Visible Web. Thus, collecting, accessing, and organizing Hidden Web resources emerges as an interesting challenge for both research and industry.

Commercial approaches to the Hidden Web have usually the form of Yahoo!-like directories pointing to local sites in specific domains. Some important examples of such directories are InvisibleWeb[1] and BrightPlanet[2], the latter also sells products that query local databases. BrightPlanet's gateway site CompletePlanet[3] is a directory as well as a metasearch engine. For each database it incorporates into its search, the metasearch engine is equipped with a manually written “wrapper”, a software component that specifies how to submit queries and extract query answers embedded into HTML-formatted result pages.

Like on the Visible Web, search resources on the Hidden Web are very heterogeneous. In particular, they vary in the document retrieval models they support (Boolean, vector-space and extended Boolean). They allow different operators for query formulation. Moreover, the syntax of supported operators can vary from one site to another. The conventional way of understanding query interface features is to do it manually. Practically speaking, that comes down to reading the help pages associated with a given search interface or probing the interface with sample queries and observing the result pages. The manual discovery of Web search interfaces has several important limitations. First, the manual approach is not scalable to the thousands of search resources that compose the Hidden Web. Second, the manual testing servers with probe queries is error-prone. Third, most large-scale search engines do not return the exact value for the number of matching documents for a query but its approximation, resulting in match numbers being noisy data. Fourth, cases of incorrect or incomplete help pages are frequent. Operators that are actually supported by an engine may not be mentioned in the help pages, and conversely, help pages might mention operators that are not supported by the engine. Finally, “impossible” match numbers are frequent. As an example, at the time of running our experiments the Google search engine states to support by default (i.e., as the meaning of a whitespace) the Boolean AND operator, meaning the match number is the number of documents containing all query terms. However, when queried with the two queries ’Java’ and ’Java AND Java’ Google reports finding 26 million and 3.5 million documents, respectively. Clearly, such a retrieval behavior does not fit the Boolean query model.

To overcome some or all limitations the manual query discovery, we address the problem of discovering the query languages of Web search servers in an automatic way. Beyond contributing to the analysis of Hidden Web resources, solving this problem will provide a basis for adding new search resources to any metasearcher (like Xerox’s askOnce [4]) in a scalable manner. This paper is organized as follows: Section 2 describes our method in detail, including the learning goal and the feature selection. In Section 3 we present the results of our experiments. Section 4 discusses related work, Section 5 gives the conclusion.

2 Experimental Methodology

To fully explore the content of the Hidden Web it is a necessary first step to understand the query interfaces and result pages that search pages provide. In this paper we attack this problem by learning whether or not certain types of queries are supported. To do that we rely on the number of matches that a query returns. This method has been successfully used in the past, e.g., in [13] to classify Web sites into a topic hierarchy. To illustrate the problems encountered when querying Web interfaces, consider the query ’Casablanca AND Bogart’. On Google, this query returns 24.500 matches (as opposed to 551.000 for 'Casablanca' and 263.000 for 'Bogart') plus the hint that the AND operator is unnecessary, because all query terms are included anyway. On the Internet Movie Database, on the other hand, the query returns 12.020 matches as opposed to only 22 for 'Casablanca' and 4 for 'Bogart'. The reason here is that 'AND' is taken literally and that all the words are implicitly OR-connected.

Our hypothesis is that it is sufficient to use a reasonably large set of search resources supporting a given feature for automatically learning the characteristic behaviors of that feature with respect to some probe queries. The method we propose for the automatic discovery is based on query-based probing of a search interface with a carefully selected set of probe queries and the analysis of the returned pages. We assume that the number of matches returned by a server for a submitted query is available on the first result page. Our approach has relatively low costs, because only a small set of servers has to be inspected manually to create a training set for the machine learning algorithms. These algorithms then produce a set of classification rules that indicate whether an operator is supported or not.

Model and approach. We target the Boolean, vector-space and extended Boolean query models and we try to cover the query features that are most frequently used by real Web resources. These include the Boolean operators AND, OR, and NOT, case sensitivity, stemming, and phrases consisting of more than one word. We note that the query feature set can be extended to address other query features, such as substrings, proximity operators, grouping, etc. We also note that when we talk about the Boolean NOT-operator, we actually mean AND-NOT. Few sites actually allow a query of type 'NOT A', because that could be used to return their complete database as the answer. Instead, it is common that sites support queries like 'A NOT B' indicating that A must be and B must not be present in matched documents, i.e., the user has to provide at least one "positive" keyword. For certain query features we consider several alternative syntaxes. For example, the Boolean operator AND may have different syntaxes; these include 'A AND B', 'A & B', '+A +B', and simply 'A B'. Like with the query features, the set of possible syntaxes for an operator is open and easily extensible for non-English resource providers. For example, a French site might use ET for the AND-operator and OU for the OR-operator. Our approach is to define certain queries, we call them *model queries*, involving one or more keywords and to investigate their semantics. For each model query we define its possible semantics. Then we use an ensemble of *probe queries* to retrieve the number of matches they have on the set of search sites. We extract the match numbers from the result pages using Xerox' iWrap toolkit [8]. Based on those numbers we learn rules that determine the semantics of the model queries.

Generating rules for discovering supported operators is not an easy task. A simple look at match numbers of probe queries might simply not work. Assume, for example, that two basic queries 'information' and 'retrieval' match 20 and 10 documents at some Web resource. If the probe query 'information AND retrieval' matches fewer documents, say 5, it does not necessarily mean that this query represents the logical AND of the two keywords. First, the query could be interpreted literally, as a phrase. Second, 'AND' could be implicitly dropped and two keywords left be processed as a phrase. An accurate detection requires a deeper analysis of other probe queries. Then, if both 'information retrieval' and 'retrieval AND information' have 5 matches as well and '"information retrieval"' has even less, say 3 matches, it is likely that the original query corresponds to the Boolean AND. As we can see, full-scale classification rules appear to have a complex 'if-then-else' structure. To automate the process of rule generation, we advocate for using machine learning techniques. To induce the classification rules we use Borgelt's decision tree toolkit [7].

Model queries. Our goal is to automatically discover the languages that Web query interfaces support. We identify nine model queries and their possible semantics as our learning goals. The model queries are in the following listed using placeholders A and B and illustrated using the keywords 'information' and 'retrieval'.

1. Model query: '*A*' (Example: '*information*'): We consider have four different semantics' for this query: LIT (the query is matched literally), CAS (the query is matched case-insensitively), SUB (any superstring of the query is also a match), and CSU (a combination of CAS and SUB).
2. Model query: *prefix(A)+'*'* (Example: '*informati**'): Here we consider two different semantics': LIT (the query is matched literally with or without the "*") and TRN (any word with the query as a prefix is a match).
3. Model query: '*A B*' (Example: '*information retrieval*'): We consider five different semantics': ADJ (the two words must appear directly in a row), AND (both words must appear in a matching document), OR (one of the words must appear in a matching document), FST (the first word must appear in a matching document), SND (the second word must appear in a matching document). In fact, "must appear" should be read as "is matched in accordance with the semantics of model query one".
4. Model query: '*"A B"*' (Example: '*"information retrieval"*'): We consider the same five semantics' for this query.
5. Model query: '*+A +B*' (Example: '*+information +retrieval*'): Again, we consider the same five semantics' here.
6. Model query: '*A AND B*' (Example: '*information AND retrieval*'): In addition to the previous five semantics' we consider the following three: ADJ3 (the three words must appear directly in a row), AND3 (all three words must appear in a matching document), and OR3 (one of the three words must appear in a matching document).
7. Model query: '*A OR B*' (Example: '*information OR retrieval*'): We consider the same eight semantics' as for the previous query.
8. Model query: '*A -B*' (Example: '*information -retrieval*'): In addition to the five semantics for the queries three, four, and five this query can have the following semantics: NOT (the latter word cannot be matched).

9. Model query: '*A NOT B*' (Example: '*information NOT retrieval*)': We consider the following nine semantics': NOT, ADJ, AND, OR, FST, SND, ADJ/3, AND/3, and OR/3.

In addition, each model query can also have the semantics UNK indicating that we were unable to determine the correct semantics from our manual inspection of the site. For the time being, we do not consider more advanced concepts such as proximity or the correction of typing errors.

Keyword selection and probe queries. To learn which operators are supported by a given site, we probe the site with probe queries and collect the numbers of matches they return. We probe the sites with queries constructed from word pairs. The word pairs fall into three categories:

1. Words that can form a phrase (such as 'information' and 'retrieval')
2. Words that don't form a phrase but are likely to occur in the same document (such as 'information' and 'knowledge')
3. Words that are unrelated (such as 'Turing' and 'wireless')

Our motivation to use word pairs from these different classes is that the numbers of matches can be very different even for the same type of query. As an example, assuming operator OR is supported in syntax OR, the query 'information OR knowledge' is likely to return only slightly more results than queries 'information' or 'knowledge' alone. On the other hand, 'Turing OR wireless' will probably return almost as many results as 'Turing' and 'wireless' combined.

We manually construct a list of word pairs for each category. For each site we use one word pair from each of the three categories. We randomly select the pairs from our lists, but we ensure that the two words have more than zero matches (otherwise, we pick a different pair). For each word pair we build a set of 17 probe queries by instantiating A and B in the 17 probe query templates, which include for example 'A', 'randomCase(A)', 'A B', '+B +A', etc.

Feature selection. An important aspect is the selection of features used for the induction. Query match numbers are raw data and cannot be directly used for building decision trees as Web resources considerably differ in size. Therefore, the query match numbers on different resources are often of different magnitude. A query may match millions of documents at Google, but only a few at a small local resource. To leverage the processing of query matches from resources of different size, we develop two different feature sets for building classifiers. In the first approach, we normalize the query matches by the maximum of matches for the two base queries 'A' and 'B'. Consequently, we obtain features with values mostly between 0 and 1 (except for queries related to the Boolean OR-operator). The second approach to the feature selection uses the "less-equal-greater" relationship between any two probe queries. We use the values 1, 0, and -1, indicating for every pair of probe queries whether the first one has more, as many, or less matches than the second one. As an example, suppose four probe queries $p_1, \ldots, p_4$ return 800, 1000, 800, and 300 matches, respectively. In our first approach they are encoded into four features $p_1^{(1)}, \ldots, p_4^{(1)}$ with the values 0.8, 1.0, 0.8, and 0.3, respectively. In our second approach they are encoded into six ($\binom{4}{2}$, that is) features $p_{1,2}^{(2)}, p_{1,3}^{(2)}, \ldots, p_{3,4}^{(2)}$ with the values -1, 0, 1, 1, 1, and 1, respectively. We learn a classifier for each of the model queries and for each of the two feature selection approaches. When a new resource is probed, each classifier takes as input the features obtained from the raw match numbers and produces a prediction on the semantics of the corresponding model query.

The sites. We picked 19 sites to run our experiments on. For the selection of the sites we employed a number of different criteria. First, the sites should represent a wide variety of domains. Second, they should also represent a wide variety of query languages. Third, they need to report the number of matches a query returns. Fourth, automatic access to the site must be possible. Practically speaking, we manually collected a list of more than a hundred sites from bookmarks, directories such as Yahoo!, or simple browsing. Then we eliminated sites in accordance with the criteria listed above. For example, many sites use standard search engines such as Inktomi[5], we eliminated most of them from the set. For the extraction of the number of matches we used Xerox' iWrap toolkit [8]. The experiments were run using the "leave-out-one"-technique. That is to say, because of our limited number of sites we ran every experiment 19 times with 18 training sites and one test site (iterating through the complete set of sites). Then we counted, how many times the test site was correctly classified.

One task that is cumbersome and error-prone is the manual classification of the sites. For each site and each operator and each syntax we need to detect whether the site supports the operator in the syntax or not. We achieve that by manually probing the sites, by evaluating the result pages, the relationships between different operators, and by checking the help pages of the site if available. Still, we faced some ambiguities. For example, on Google, the query 'information' returns 195.000.000 matches, 'retrieval' returns 2.110.000 matches. Now, 'information -retrieval' returns 3.720.000. If model query 8 had semantics NOT, then the last query should return at least close to 193.000.000 matches. However, we know that Google's results are approximations of the real match numbers, and other word combinations give more reasonable results. Also, the help pages suggest that this operator is supported in the given syntax. So we decided to classify model query 8 for Google as NOT.

Summary. Figure 1 summarizes our approach. In the learning phase we send for each learning goal (model query) probe queries P_j composed of certain keywords to some training sites and collect the match numbers N_j they return. The training sites are manually classified with respect to the possible semantics C_i of the model query, which then allows us to build a decision tree. That tree can determine the semantics of the model query based on the match numbers of the probe queries. In the application phase a new site to be classified automatically is probed with the same probe queries (possibly using different keywords). The match numbers serve as input to the decision tree, which produces the classification C of the site.

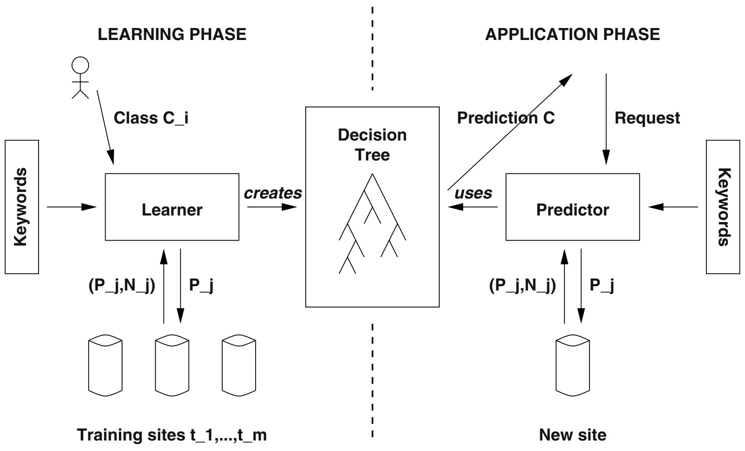


Fig. 1. Summary of our overall approach

3 Experimental Results

In this section we report our experimental results. They are summarized in Table 1. The rows list the results for the different model queries. The second column lists the result for the basic approach as described before. The percentage values represent the numbers of test runs (out of the total of 19 for each model query) where the test site has been classified correctly. (Recall that we employed the "leave-one-out"-technique.) As can be seen, for some of the model queries the results are quite good. Based on the experiences we got from the manual classification of the sites we didn't expect to achieve much more than 80% accuracy. However, for the more complex model queries such as 7 and 9, which have more alternatives for classification, the results are not optimal.

We decided to adopt more intelligent feature selection strategies. Our first strategy is called "Reduced Feature Set (RFS)". Here we reduced the set of probe queries to cope with the "feature overload" and the resulting possibility of overfitting. In particular, for each model query we selected only those probe queries (out of the total set of 17 probe queries) that we thought were necessary for that specific model query (only two probe queries for model query 2, eight probe queries for the most complex model queries 8 and 9). Our second strategy is called "Background Knowledge (BK)". There are obviously dependencies between the model queries. If model query '*A B*' has semantics AND, it is unlikely that model query '*A AND B*' has semantics OR3. So we impose an order on the learning goals (model queries) and incorporate the resulting site classifications of already processed ("earlier") model queries into the feature set of the remaining ("later") model queries (in reality, we use the "correct" manual site classifications instead of the automatically generated, but possibly incorrect ones). Finally, because these two strategies are independent from each other, we combined them resulting in the strategy RFS + BK.

Table 1. Results of the experiments

Model query	Basic	RFS	BK	RFS + BK	Manual rules
1 ('A')	95%	100%	95%	100%	90%
2 (prefix(A)+'*')	89%	84%	89%	84%	93%
3 ('A B')	74%	84%	74%	84%	90%
4 ('"A B"')	63%	79%	63%	74%	93%
5 ('+A +B')	42%	63%	47%	74%	86%
6 ('A AND B')	74%	79%	74%	79%	92%
7 ('A OR B')	47%	53%	47%	58%	80%
8 ('A -B')	74%	84%	74%	84%	94%
9 ('A NOT B')	58%	58%	58%	58%	74%

Table 1 lists in columns three, four, and five the results we got for our different feature selection strategies. It can be observed that in particular the RFS strategy is effective. When applied by itself the results are already significantly better, and when combined with BK, there is even further improvement. Still, for the model queries 7 and 9 the success rate is below 60%. One reason we see here is that these queries also have the highest number of possible classifications, so there is not enough training data for each of the possible cases and "guessing" in case of doubt is likely to be unsuccessful.

Based on the experiences we got from the manual classification of sites we generated classification rules for each of the model queries by hand. The results are listed in the sixth column of Table 1. We can see that these rules produce the best results (except for model query 1). So one could wonder, why we do not simply use those rules instead of the ones generated by the machine learning techniques. The reason here is that they are difficult to maintain. It was a very cumbersome task to write them, but any change in retrieval models, query features, syntaxes, etc. would possibly require a complete rewrite. We rather view the results we obtained with these rules as an upper bound of what is realistically achievable.

One could also wonder why these manual results do not perform perfectly. After all, they are the basis of the manual classification we performed. First, there is again the issue of noise in data. The manual classification involved more than just sending a fixed set of probe queries. In doubt, we could use other queries, other keywords, etc. Also, we could make use of the help pages provided by a site. Second, there is the question of encoding. The "less-equal-greater"-encoding is sometimes not powerful enough. If the probe query 'information retrieval' returns more matches than either 'information' or 'retrieval' the manual rules classify its semantics as a model query as OR. However, if the real numbers are something like 318.490, 57.263, and 3.929 respectively, as on www.ieee.org some time ago, we have to manually classify that semantics as UNK.

4 Related Work

Research around the Hidden Web is just emerging. A general overview of its characteristics in terms of size and quality can be found in [6]. One focus of the research is crawling: in [15] a crawling approach is presented that tries to index pages from the Hidden Web. This work is complementary to ours, because HTML forms are filled with entries from a dynamically managed set. Classification of Web resources is another important task. [13] and [16] demonstrate approaches to this problem based on probing. Incidentally, the former of the two makes use of the number of documents matching a submitted query, as does our approach (alas for a different purpose). Related to classification is the problem of database selection in metasearching. [10], it's follow-up [9], and [12] address this problem. The former two construct language models from the results of query probing, the latter again makes use of the match numbers of some one-word queries. In [14], interaction with online vendors is automated. This is another type of complementary research that could benefit from our work. In the domain of metasearching, the declaration of a search resource's query features is often coupled with methods of converting/translating metasearch queries into the resource's native queries. A considerable research effort has been devoted to minimizing possible overhead of query translation when the metasearch and search resource differ in supporting basic query features [11]. In all these methods, the manual discovery the resource's query features is assumed.

5 Discussion and Conclusion

In this paper we describe how to automatically identify query language features of Web search interfaces using machine learning techniques. Our approach is based on the number of matches a server returns for a given query. There are several aspects that make this a difficult problem. There is the aspect of the retrieval model (Boolean, vector-space, or extended Boolean). Match numbers are often estimates. Help pages are incorrect or incomplete. We address the problem by manually classifying a set of training sites and learning rules on how to classify a new site given the match numbers of some sample queries. Our contribution of this paper is useful not only for exploring the "Hidden Web" but for metasearching in general.

Our work is part of a more broader project to enable automatic discovery of the Hidden Web. We believe that three main tasks are to be addressed here. First, there is discovery. We need crawlers that can identify potential Hidden Web information providers. Second, there is analysis. These potential providers need to be understood. How can they be queried? What kind of answers do they return? How can these answers be processed? The work of this paper falls into this second task. Third, there is classification. Because of the vastness of the Web, information providers need to be classified into some predefined hierarchy of categories.

In the future we will attempt to improve our results by trying out various new ideas. A simple idea is to use machine learning techniques other than decision tree, for example the k-nearest neighbor algorithm or support vector machines (SVM's). Another idea is to reformulate the goal of learning. Finally, we are looking into extending our work to cover complex queries and attribute searches.

References

1. The InvisibleWeb, http://www.invisibleweb.com/.
2. BrightPlanet, http://www.brightplanet.com/.
3. CompletePlanet, http://www.completeplanet.com/.
4. askOnce: The Enterprise Content Integration Solution, http://www.askonce.com/.
5. Inktomi, http://www.inktomi.com/.
6. M. K. Bergman. The Deep Web: Surfacing hidden value. *Journal of Electronic Publishing*, 7(1), 2001.
7. C. Borgelt. Christian Borgelt's software page. http://fuzzy.cs.uni-magdeburg.de/ borgelt/software.html.
8. D. Bredelet and B. Roustant. Java IWrap: Wrapper induction by grammar learning. Master's thesis, ENSIMAG Grenoble, 2000.
9. J. Callan and M. Connell. Query-based sampling of text databases. *ACM Transactions on Information Systems (TOIS)*, 19(2):97–130, 2001.
10. J. P. Callan, M. Connell, and A. Du. Automatic discovery of language models for text databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 479–490, Philadelphia, PA, USA, June 1999.
11. C.-C. K. Chang, H. Garcia-Molina, and A. Paepcke. Boolean query mapping across heterogeneous information sources. *IEEE Transactions on Knowledge and Data Engineering*, 8(4):515–521, 1996.
12. P. G. Ipeirotis and L. Gravano. Distributed search over the hidden web: Hierarchical database sampling and selection. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 394–405, Hong Kong, China, August 2002.
13. P. G. Ipeirotis, L. Gravano, and M. Sahami. Probe, count, and classify: Categorizing hidden-web databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 67–78, Santa Barbara, CA, USA, May 2001.
14. M. Perkowitz, R. B. Doorenbos, O. Etzioni, and D. S. Weld. Learning to understand information on the internet: An example-based approach. *Journal of Intelligent Information Systems*, 8(2):133–153, 1997.
15. S. Raghavan and H. Garcia-Molina. Crawling the hidden web. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 129–138, Rome, Italy, September 2001.
16. W. Wang, W. Meng, and C. Yu. Concept hierarchy based text database categorization. In *Proceedings of the International Conference on Web Information Systems Engineering (WISE)*, pages 283–290, Hong Kong, China, June 2000.

The Effect of Database Size Distribution on Resource Selection Algorithms

Luo Si and Jamie Callan

School of Computer Science, Carnegie Mellon University.
5000 Forbes Ave, Pittsburgh PA, U.S.A.
{lsi, callan}@cs.cmu.edu

Abstract. Resource selection is an important topic in distributed information retrieval research. It can be a component of a distributed information retrieval task and can also serve as an independent application of database recommendation system together with the resource representation part. There is a large body of valuable prior research on resource selection but very little has studied about the effects of different database size distributions on resource selection. In this paper, we propose extended versions of two well-known resource selection algorithms: CORI and KL divergence in order to consider the factors of database size distributions, and compare them with the lately proposed Relevant Document Distribution Estimation (ReDDE) resource selection algorithm. Experiments were done on four testbeds with different characteristics, and the ReDDE and the extended KL divergence resource selection algorithm have been shown to be more robust in various environments.

1 Introduction

The proliferation of online searchable databases on local area networks and the Internet has increased attention on the problem of finding information that may be distributed among many text databases (*distributed information retrieval*) [1]. There are three important sub-problems in distributed information retrieval: first, the content of each text database must be represented in a suitable form (resource representation); second, given an information need (a query), several relevant databases must be selected to do the search (resource selection); and third, the results from searching the selected databases have to be merged into a single final list (result merging) [1].

This paper focuses on resource selection. Many valuable resource selection algorithms have been proposed and compared in the literature [1, 2, 3, 8, 9], but no thorough study has been conducted to study the performance of different algorithms in environments with different database size distributions. Two well-known algorithms, CORI [1, 2] and KL divergence [7, 9] resource selection, were evaluated and extended to explicitly incorporate database size factors. Four testbeds with different characteristics were created. The CORI and the KL divergence algorithms (and their extensions) were compared to the recently-proposed ReDDE algorithm [8]. The experiments support two conclusions. First, in order to have good performance in different environments, it is very important to include a database size factor into a resource selection algorithm. Second, the ReDDE and the extended KL divergence algorithms are the better and more stable algorithms among the algorithms considered in this research.

The rest of this paper is organized as follows. Section 2 discusses previous research. Section 3 describes the proposed extensions to the CORI and KL divergence resource selection algorithms; it also describes the ReDDE algorithm. Section 4 discusses our experimental methodology. Section 5 presents and analyzes experimental results. Section 6 concludes and discusses future work.

2 Previous Work

There has been considerable research on acquiring resource representation, resource selection, and merging results. As this paper is focused on resource selection algorithm, we mainly survey related work in resource representation and resource selection in this section, and briefly introduce the work of results merging.

2.1 Resource Representation

The STARTS protocol is one solution for acquiring resource descriptions [4]. It needs the cooperation from every resource provider to provide accurate resource descriptions. STARTS is a good solution in environments where cooperation can be guaranteed. However, it does not work in multi-party environments where some resource providers may not cooperate.

Query-based sampling [1, 2] is an alternative approach to acquiring resource descriptions that does not require explicit cooperation from resource providers. Query-based sampling only asks the search engines to run queries and return a list of downloadable documents. In practice, this solution has been shown to acquire rather accurate resource descriptions using a relatively small number of randomly-generated queries (e.g., 75) to retrieve a relatively small number of documents (e.g., 300).

In order for a resource selection algorithm to use database size information each resource representation must contain database size statistics. In cooperative environments each database may cooperate to provide this information. In uncooperative environments learning algorithms must be applied to estimate database sizes. The capture-recapture [5] and the sample-resample algorithms [8] have been proposed to fulfill this task. In the recent work [8], the sample-resample algorithm was shown to acquire relatively accurate database size estimates efficiently and robustly in different environments. It can be extremely efficient when query-based sampling is used to create resource descriptions [8].

2.2 Resource Selection

The goal of resource selection is to select a small set of resources that contain a lot of relevant documents. There are many successful resource selection algorithms, for example, CORI [1, 2], KL divergence [7, 9] and ReDDE [8].

The CORI algorithm uses a Bayesian inference network and an adaptation of the Okapi term frequency normalization formula to rank resources. Prior research by several different researchers using different datasets has shown the CORI algorithm to be one of the most stable and effective algorithms [1, 3]. The belief $p(r_k \mid c_i)$ in database db_i according to the query term r_k is determined by:

$$T = \frac{df}{df + 50 + 150 * cw_i / avg\_cw} \tag{1}$$

$$I = \frac{\log(\frac{|DB| + 0.5}{cf})}{\log(|DB| + 1.0)} \tag{2}$$

$$p(r_k \mid c_i) = b + (1 - b) * T * I \tag{3}$$

where: df is the number of documents in db_i that contain r_k;
cf is the number of databases that contain r_k;
$|DB|$ is the number of databases to be ranked;
cw_i is the number of words in db_i;
avg_cw is the average cw of the databases to be ranked; and
b is the default belief, usually set to 0.4.

The Kullback-Leibler (KL) divergence collection selection method was proposed by Xu and Croft [9]. In their work, the Kullback-Leibler divergence between the word frequency distribution of the query and the database is used to measure how well the content of the database matches with the query. This algorithm can be represented in the language modeling framework as [7]. The algorithm and an extension are described in the next section.

The database size normalization components of the CORI and KL-divergence algorithms are weak, at best. CORI represents database size by the number of words the database contains as compared with a mean (cw_i and *avg_cw*). The KL-divergence algorithm has no explicit normalization for database size. Indeed, although much valuable research has studied the performance and robustness of resource selection algorithms [3, 8, 9], very little of it has done evaluated the effects of different database size distributions on resource selection.

The ReDDE resource selection algorithm [8] is a newly proposed algorithm that explicitly tries to estimate the distribution of relevant documents across the set of available databases. The ReDDE algorithm considers both content similarity and database size when making its estimates (see Section 3).

2.3 Result Merging

The database representation and the resource selection components can be combined to create a database recommendation system that suggests which online databases are the best for a query. If the user wants the system to further search the selected databases and merge the returned results from each database, a result-merging component is needed. For example, the Semi Supervised Learning (SSL) [6] result merging algorithm uses the documents acquired by query-based sampling as training data and linear regression to learn the database-specific, query-specific merging models.

3 Resource Selection Algorithms That Normalize for Database Size

In this section, we extend the CORI and KL-divergence resource selection algorithms to normalize for database sizes; we also describe the ReDDE algorithm.

3.1 The Extension of the CORI Resource Selection Algorithm

Callan [1] pointed out that Equation 1 in the CORI resource selection algorithm is a variation of Roberson's term frequency (tf) weight, in which term frequency (tf) is replaced by document frequency (df), and the constants are scaled by a factor of 100 to accommodate the larger df values. We generalize Equation 1 to create Equation 4 as follows:

$$T = \frac{df}{df + df\_base + df\_factor * cw_i / avg\_cw} \tag{4}$$

where df_base and df_factor represents the two constants. There are three issues in the above formula that should be addressed to incorporate the database size factor.

First, df in Equation 4 is the number of sampled documents from the ith database that contain r_k;. In order to estimate the actual number of corresponding documents in the database, it should be scaled as:

$$df' = \frac{df}{N_{C_i\_samp}} * \hat{N}_{C_i} \tag{5}$$

where $N_{C_i\_samp}$ is the number of documents sampled from the database, and $\hat{N}_{C_i}$ is the estimated number of documents in the database.

Second, the cw_i in Equation 4 is the estimated number of words in the ith database, which is calculated from the documents sampled from the database. In order to consider the database size, this number should also be scaled. Formally as:

$$cw_i' = \frac{1}{N_{C_i\_samp}} * \hat{N}_{C_i} * cw_i \tag{6}$$

Taking a similar approach to avg_cw in Equation 4 yields:

$$avg\_cw' = \frac{1}{C} \sum_i \frac{1}{N_{C_i\_samp}} * \hat{N}_{C_i} * cw_i \tag{7}$$

Finally, just as the original CORI algorithm adjusts the df_base and **df**_factor constants by a factor of 100 to accommodate the larger df values (among several hundred sampled documents), we scale the df_base and df_factor constants as shown in Equations 8 and 9 to compensate for variations in database size.

$$df\_base' = \frac{\hat{N}_{C_i}}{N_{C_i\_samp}} * 50 \tag{8}$$

$$df\_factor' = \frac{\hat{N}_{C_i}}{N_{C_i\_samp}} * 150 \tag{9}$$

In this work, we make two extensions of the CORI algorithm to address these two issues. The first extension, which we call CORI_ext1, uses the new document frequency, the new collection word count and the new average collection word count in Equation 5, 6 and 7. The second extension, which we call CORI_ext2, leverages all the new document frequency, the new collection word count (average count) and the new df_base and df_factor constants.

3.2 The Extension of the KL-Divergence Resource Selection Algorithm

The KL-divergence resource selection algorithm has been given an interpretation in the language-modeling framework in [7]. In this framework, the documents sampled from a database are collapsed into one single, giant 'document' and used in the computation of the collection-query similarity. More formally, we need to find the collections that have largest probabilities of P (Ci | Q), i.e. the conditional probability of predicting the ith database given the query. By the Bayesian rule, P (Ci | Q) can be calculated as:

$$P(C_i \mid Q) = \frac{P(Q \mid C_i) * P(C_i)}{P(Q)} \tag{10}$$

where P (Q | Ci) denotes the probability of generating the query Q from the ith database. P(Ci) is the prior probability of the specific database. P(Q) is the generation probability of the query, which is database independent and can be neglected. Collections with the largest generation probabilities P (Ci | Q) are selected.

Following the principle of a unigram language model with independent terms, the value of P (Q | Ci) is calculated as:

$$P(Q \mid C_i) = \prod_{q \in Q} \left(\lambda P(q \mid C_i) + (1-\lambda) P(q \mid G) \right) \tag{11}$$

where q is a query term. P (q | Ci) is the language model for the ith database and P (q | G) is the language model for the whole collection. The linear interpolation constant λ smoothes the collection-based language model with the global language model, which was set to 0.5 in our experiments.

The only item left in Equation 10 that needs to be estimated is the prior probability P(Ci). If it is assigned as a uniform distribution, the language model resource selection algorithm can be shown by simple mathematical manipulation to be equal to the original KL-divergence algorithm [7, 9].

The natural way to introduce a database size factor into the KL-divergence resource selection algorithm is to assign the database prior probabilities according to the database size. Formally as:

$$P(C_i) = \frac{\hat{N}_{C_i}}{\sum_j \hat{N}_{C_j}} \tag{12}$$

where the denominator is the estimated total testbed size. Plugging Equation 11 and 12 into Equation 10, we get a new version of the KL-divergence resource selection algorithm, which we call the KL_ext algorithm.

3.3 The ReDDE Resource Selection Algorithm

The newly proposed ReDDE (Relevant Document Distribution Estimation) algorithm explicitly estimates the distribution of relevant documents across all the databases. It considers both content similarity and database size when making its estimates.

The number of documents relevant to query Q in the ith database Ci is estimated as:

$$\text{Re}\hat{l}\_Q(i) = \sum_{d_j \in C_{i}\_samp} P(rel \mid d_j) * \frac{1}{N_{C_i\_samp}} * \hat{N}_{C_i} \tag{13}$$

Again, $N_{C_i\_samp}$ is the number of sampled documents from the ith database and $\hat{N}_{C_i}$ is the estimated size of this database. P (rel | dj) is the probability of relevance given a specific document. The probability is calculated by reference to the *centralized complete database*, which is the union of all of the individual databases available in the distributed IR environment. We simplify P (rel | dj) as the probability of relevance given the document rank when the centralized complete database is searched by an effective retrieval method. The probability distribution function is a step function, which means that for the documents at the top of the ranked list the probabilities are a positive constant, and are 0 otherwse. This can be modeled as follows:

$$P(rel \mid d_j) = \begin{cases} C_Q & \text{if } Rank\_central(d_j) < ratio * \sum_i \hat{N}_{C_i} \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

The centralized complete database is not accessible, but the rank of a document in the centralized complete database can be estimated, as shown below, from its rank within the *centralized sample database,* which is the union of all the sampled documents from different databases.

$$Rank\_Central(d_j) = \sum_{\substack{d_k \\ Rank\_Samp(d_k) < Rank\_Samp(d_j)}} \frac{\hat{N}_{c(d_k)}}{N_{c(d_k)\_samp}} \tag{15}$$

The last step is to normalize the estimates in Equation 13 to remove the query dependent constants, which provides the distribution of relevant documents among the databases.

$$Dist\_\hat{Rel}\_Q(i) = \frac{\hat{Rel}\_Q(i)}{\sum_j \hat{Rel}\_Q(j)} \tag{16}$$

ReDDE selects the databases with the highest mass in the distribution.

4 Experimental Methodology

Two well-known testbeds were used in our experiments.

1. **Trec123-100col-bysource:** 100 databases were created from TREC CDs 1, 2 and 3. They were organized by source and publication date [1]. The sizes of the databases are not skewed. This testbed has been used often in prior research [1, 7, 8].
2. **Trec4-kmeans:** 100 databases were created from TREC 4 data. A k-means clustering algorithm was used to organize the databases by topic [9], so the databases are homogenous and the word distributions are very skewed. The sizes of the databases are moderately skewed.

The characteristics of these two testbeds are shown in Table 1.

To investigate the effect of various database size distributions on resource selection algorithms, two new testbeds were created based on the Trec123-100col testbed.

Table 1. Summary statistics for the Trec123-100colbysource and Trec4-kmeans testbeds.

Name	Query Count	Size (GB)	Number of Docs			Megabytes (MB)		
			Min	Avg	Max	Min	Avg	Max
Trec123	50	3.2	752	10782	39713	28.1	32	41.8
Trec4_kmeans	50	2.0	301	5675	82727	3.9	20	248.6

1. **Trec123-AP-WSJ-60col ("relevant"):** The 24 Associated Press collections in the trec123-100col-bysource testbed were collapsed into a single large APall database. The 16 Wall Street Journal collections were collapsed into a single large WSJall collection. Details are shown in Table 2. The other 60 collections were left unchanged. The APall and WSJall collections are much larger than the other 60 databases, and they also have a higher density of documents relevant to TREC queries than the other collections.
2. **Trec123-FR-DOE-81col ("nonrelevat"):** The 13 Federal Register collections and the 6 Department of Energy collections in the trec123-100col-bysource testbed were collapsed into two large collections FRall and DOEall respectively. Details are shown in Table 2. The other 81 collections were left unchanged. The FRall and DOEall databases are much larger than the other databases, but have a much lower density of relevant documents than other databases.

Table 2. Summary statistics for the large databases.

Collection	Num of Docs	Size (MB)
APall	242,918	764.3
WSJall	173,252	533.2
FRall	45,820	491.6
DOEall	226,087	192.7

These two testbeds were created to simulate environments where large collections contain i) a large percentage of relevant documents, and ii) a small percentage of relevant documents.

50 short queries were created from the title fields of TREC topics 51-100 for the Trec123-100bycol, relevant and nonrelevant testbeds. 50 longer queries were created from the description fields of TREC topics 201-250 for the Trec4-kmeans testbed.

5 Experimental Results

The experiments were done on 4 testbeds to evaluate 6 resource selection algorithms, namely CORI, CORI_ext1, CORI_ext2, KL divergence, KL_ext and the ReDDE algorithms. The testbeds cover a wide range of conditions: relatively uniform sizes

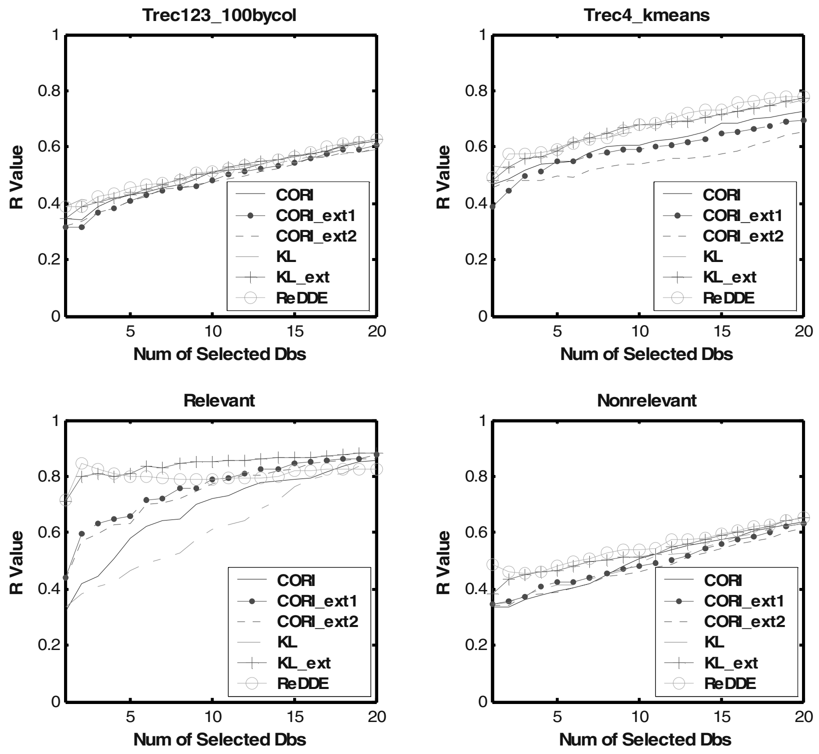


Fig. 1. The performance of the resource selection algorithms with estimated database sizes

and content distribution (trec123-100col), relatively uniform sizes and skewed content distribution (trec4-kmeans), bimodal size distribution and a much higher density of relevant documents in the large databases (relevant), and bimodal size distribution and a much lower density of relevant documents in the large databases (nonrelevant).

Query based sampling was used to acquire the database resource descriptions by repeatedly sending one-term queries to databases and downloading the top 4 documents returned until 300 unique documents were downloaded. The sample-resample method was used to estimate database sizes.

Resource selection algorithms are typically compared using the recall metric R_n [1, 3, 8]. Let us denote B as a baseline ranking, which is often the RBR (relevance based ranking), and E as a ranking provided by a resource selection algorithm. And let B_i and E_i denote the number of relevant documents in the ith ranked database of B or E. Then R_n is defined as shown below.

$$R_k = \frac{\sum_{i=1}^{k} E_i}{\sum_{i=1}^{k} B_i} \tag{16}$$

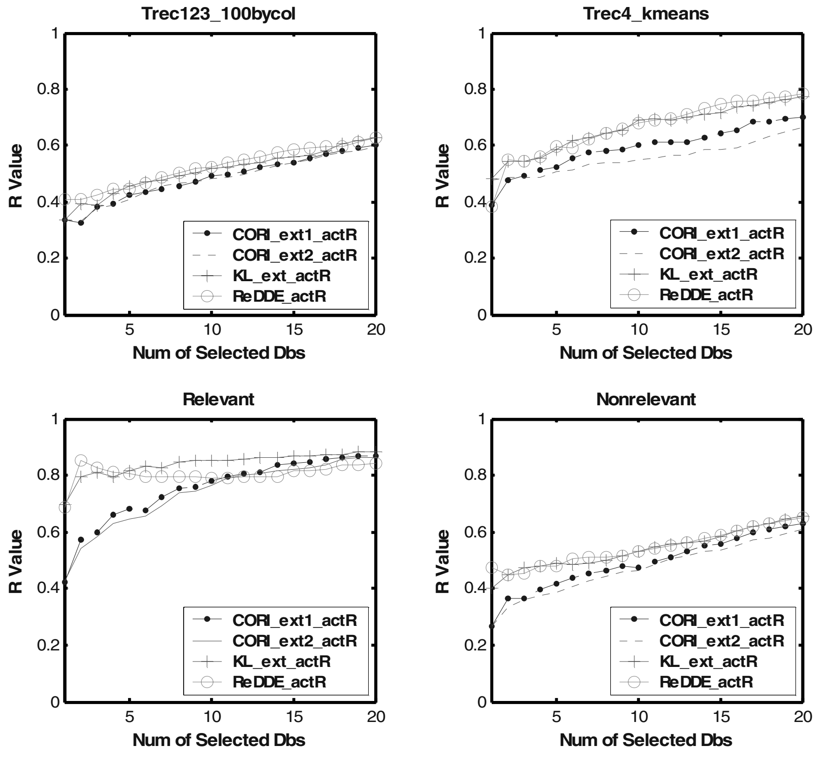


Fig. 2. The performance of the resource selection algorithms with actual database size ratios

Usually the goal is to search only a few databases, so in our experiments the algorithms were only evaluated over the top 20 databases.

Two sets of experiments were done to study the performance of different resource selection algorithms with either estimated database sizes (Figure 1) or actual database sizes (Figure 2).

For the experiments with estimated database sizes, the resource selection algorithms that do not consider the database size factor (CORI and KL-divergence algorithm) do reasonably well on the testbeds with normal or modestly skewed database size distributions (trec123_100bycol and the trec4_kmeans testbed), but their performance on the testbeds with very skewed database size distributions (relevant and nonrelevant testbeds) are not satisfactory. This suggests that the database size distribution is a very important factor for robust resource selection algorithms that must work in a wide range of environments.

The two extensions of the CORI resource algorithms (CORI_ext1 and CORI_ext2) have inconsistent performance on the four testbeds. Sometimes they are better than the original version (relevant testbed), and sometimes they are even worse than the original algorithm (trec4_kmeans testbed). This does not mean there is no other good approach to modifying the CORI algorithm; it just indicates that the two extensions proposed in this paper are not robust.

The extension of the KL-divergence algorithm (KL_ext) and the ReDDE algorithm had consistently good performance on all of the four testbed. We conclude that the database size normalizations in the ReDDE algorithm and the extension of the KL-divergence resource algorithm are effective, and that normalizing for database sizes is important to obtaining robust behavior when database sizes are skewed.

In Figure 2, the resource selection algorithms that incorporate database size factors were tried to evaluate their behavior with actual database sizes instead of estimated sizes. The experiments have shown consistent result with what we get from Figure 1, which is that the extension of the KL-divergence algorithm (KL_ext_act) and the ReDDE algorithm are more effective and robust than the CORI extensions on all the testbeds.

6 Conclusions and Future Work

Resource selection is a hot research topic in distributed information retrieval and many valuable algorithms have been studied. However, to our knowledge the effects of database size distributions on different resource selection algorithms has not been evaluated thoroughly. In this work, we propose extensions of two well-known algorithms: CORI and KL-divergence. We evaluated original and extended versions of several resource selection algorithms on four testbeds that have different characteristics. The experiments show that it is important to include a database size factor into resource selection algorithms; the ReDDE and extended KL-divergence algorithms had the best and most stable performance among the algorithms evaluated.

By carefully studying the experiment results, it can be seen that the advantage of the ReDDE algorithm when selecting more than 10 databases is smaller than when selecting a few databases (e.g. relevant testbed). We attribute this to the simplification of Equation 14 with a step function. We believe that the function can be estimated by using training data to improve the accuracy in the future work.

References

1. Callan, J.: Distributed information retrieval. In: Croft, W.B. (eds.): Advances in Information Retrieval. Kluwer Academic Publishers (2000) 127–150
2. Callan, J., Connell, M.: Query-based sampling of text databases. ACM Transactions on Information Systems (2001) 97–130
3. French, J.C., Powell, A.L., Callan, J., Viles, C.L., Emmitt, T., Prey, K.J., Mou, Y.: Comparing the performance of database selection algorithms. In Proceedings of the Twenty Second Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (1999)

4. Gravano, L., Chang, C., Garcia-Molina, H., Paepcke, A.: STARTS: Stanford proposal for internet Meta-Searching. In Proceedings of the ACM-SIGMOD International Conference on Management of Data (1997)
5. Liu, K.L., Yu, C., Meng, W., Santos, A., Zhang, C.: Discovering the representative of a search engine. In Proceedings of 10th ACM International Conference on Information and Knowledge Management (2001)
6. Si, L., Callan, J.: Using sampled data and regression to merge search engine results. In Proceedings of the Twenty Fourth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (2002)
7. Si, L., Jin, R., Callan, J., Ogilvie, P.: A language model framework for resource selection and results merging. In Proceedings of the eleventh International Conference on Information and Knowledge Management, ACM (2002)
8. Si, L., Callan, J.: Relevant document distribution estimation method for resource selection. In Proceedings of the Twenty Fifth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (2003)
9. Xu, J., Croft, W.B.: Cluster-based language models for distributed retrieval. In Proceedings of the Twenty Second Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (1999)

Decision-Theoretic Resource Selection for Different Data Types in MIND

Henrik Nottelmann and Norbert Fuhr

Institute of Informatics and Interactive Systems, University of Duisburg-Essen, 47048 Duisburg, Germany, {nottelmann,fuhr}@uni-duisburg.de

Abstract. In a federated digital library system, it is too expensive to query every accessible library. Resource selection is the task to decide to which libraries a query should be routed. In this paper, we describe a novel technique that is used in the MIND project. Our approach, decision-theoretic framework (DTF), differs from existing algorithms like CORI in two ways: It computes a selection which minimises the overall costs (e.g. retrieval quality, time, money) of the distributed retrieval. And it allows for other data types beside text (e.g., names, years, images), whereas other resource selection techniques are restricted to text.

1 Introduction

Resource selection is the task to determine automatically useful ("relevant") collection in a federation of digital libraries (DLs). Traditional algorithms (e.g. GlOSS, CORI) compute similarities between the library and the query, and retrieve a constant number of documents from the top-ranked libraries.

The GlOSS system [8] is based on the vector space model and – thus – does not refer to the concept of relevance. For each library, a goodness measure is computed which is the sum of all (SMART) scores of all documents in this library w. r. t. the current query. Libraries are ranked according to the goodness values.

The state-of-the-art system CORI [3] uses the INQUERY retrieval system which is based on inference networks. The resource selection task is reduced to a document retrieval task, where a "document" is the concatenation of all documents of one library. The indexing weighting scheme is quite similar to the DTF one, but applied to libraries instead of documents. Thus, term frequencies are replaced by document frequencies, and document frequencies by collection frequencies. CORI also covers the data fusion problem, where the library score is used to normalise the document score. CORI is one of the best performing resource selections models.

In contrast, the decision-theoretic framework (DTF) [7,13] has a better theoretic foundation: The task is to find the selection with minimum costs (which depend on different criteria like retrieval quality, time or money). A user can choose different selection policies by specifying the importance of the different cost sources. Three different methods for estimating retrieval quality for text libraries have been developed for DTF so far.

In this paper, we extend DTF towards additional data types (like names, years or images) and search predicates (e.g. `sounds-like` for names or `=` for years), which have to be handled slightly different.

We implemented this new method within the MIND project. MIND is a federated digital library system for non-co-operating and multi-media DLs.

The rest of this paper is organised as follows. In the next section, we introduce the decision-theoretic framework for resource selection. In Sect. 3, we describe the retrieval model used in DTF, and how different data types (text, names, years, images) are integrated. Different methods for estimating retrieval quality are presented in Sect. 4. Section 5 contains some evaluation results for text. The last section contains some concluding remarks and an outlook to future work.

2 Decision-Theoretic Framework

The basic assumption of the decision-theoretic framework (DTF) [7,13] is that we can assign specific retrieval costs $C_i(s_i, q)$ to each digital library DL_i, where s_i is the number of documents retrieved for query q. The term "costs" is used in a broad way and also includes—besides money—cost factors like time and quality.

If the user specifies (together with her query) the total number n of documents which should be retrieved, the task then is to compute an optimum solution, i.e. a vector $\boldsymbol{s} = (s_1, s_2, \ldots, s_m)^T$ with $|\boldsymbol{s}| = \sum_{i=1}^{m} s_i = n$ which minimises the overall costs:

$$M(n, q) := \min_{|\boldsymbol{s}|=n} \sum_{i=1}^{m} C_i(s_i, q). \tag{1}$$

For $C_i(s_i, q)$, costs from different sources should be considered:

Effectiveness: Probably most important, a user is interested in getting many relevant documents. Thus we assign user-specific costs C^+ for viewing a relevant document and costs $C^- > C^+$ for viewing an irrelevant document. If $r_i(s_i, q)$ denotes the number of relevant documents in the result set when s_i documents are retrieved from library DL_i for query q, we obtain the cost function:

$$C_i^{rel}(s_i, q) := r_i(s_i, q) \cdot C^+ + [s_i - r_i(s_i, q)] \cdot C^-. \tag{2}$$

Time: This includes computation time at the library and communication time for delivering the result documents over the network. These costs can easily be approximated by measuring the response time for several queries. In most cases, a simple affine linear cost function is sufficient:

$$C_i^{time}(s_i, q) := C_0^{time} + C_1^{time} \cdot s_i. \tag{3}$$

Money: Some DLs charge for their usage, and monetary costs often are very important for a user. These costs have to be specified manually. In most cases, the cost function is purely linear (reflecting per-document-charges):

$$C_i^{money}(s_i, q) := C^{money} \cdot s_i. \tag{4}$$

Thus, in most cases we have $C^{money} = 0$.

By summing up the costs from different sources, we arrive at an overall cost function $C_i(s_i, q)$ with user-defined cost parameters C^+ and C^- (for effectiveness), C_0^{time} and C_1^{time} (for time), and C^{money} (for money):

$$C_i(s_i, q) = C_i^{rel}(s_i, q) + C_i^{time}(s_i, q) + C_i^{money}(s_i, q). \tag{5}$$

Thus, a user can specify her own selection policy (e.g. cheap and fast results with a potentially smaller number of relevant documents). But as we do not know the actual costs (particularly the number of relevant documents) in advance, we switch to expected costs $EC_i(s_i, q)$ (for relevancy costs, using the expected number $E[r_i(s_i, q)]$ of relevant documents):

$$EM(n, q) := \min_{|\boldsymbol{s}|=n} \sum_{i=1}^{m} EC_i(s_i, q), \tag{6}$$

$$EC_i(s_i, q) := EC_i^{rel}(s_i, q) + EC_i^{time}(s_i, q) + EC_i^{money}(s_i, q). \tag{7}$$

In (6), the expected costs $EC_i(s_i, q)$ are increasing with the number s_i of documents retrieved. Thus, the algorithm presented in [7] can be used for computing an optimum solution.

3 The MIND Retrieval Model

In this section we describe the underlying retrieval model and the integration of different data types.

3.1 General Model

The retrieval model follows Risjbergen's [18] paradigm of IR as uncertain inference, a generalisation of the logical view on databases. In uncertain inference, IR means estimating the probability $Pr(q \leftarrow d)$ that the document d logically implies the query q ("probability of inference").

In MIND, documents adhere to a schema which defines their structure. In this paper, we simply assume that there is only one single schema; a method for dealing with heterogeneous schemas is presented in [12]. Schemas are built on top of data types. A data type D defines the set of possible values in the documents $dom(D)$ (the "domain") and a set of search predicates $pred(D) = \{p_1, \ldots, p_n\}$. Each predicate is a function $p_i : dom(D) \times dom(D, p) \to [0, 1]$, where $dom(D, p_i)$ is the set of all possible comparison values w. r. t. this search predicate p_i. Examples of data types are presented in the remainder of this section.

MIND employs a linear document model. A schema S is a set of schema attributes, and each schema attribute $A \in S$ belongs to a specific data type $dt(A)$:

$$\begin{aligned} S_1 := \{&au = (Name, \{=, sounds - like\}),\\ &da = (DateISO8601, \{=\}),\\ &py = (Year, \{=, <, <=, >, >=, \sim<, \sim>, \sim=\}),\\ &ti = (Text, \{contains, containsNoStem\})\}. \end{aligned}$$

Analogously, a document d contains a set of document attributes, where a document attribute is a pair $(A, v(A))$ of a schema attribute $A \in S$ and a document value $v(A) \in dom(dt(A))$:

$$d := \{(au, C.J.\ Rijsbergen), (ti, Probabilistic\ Retrieval\ Revisited)\}.$$

A query q (adhering to the schema S) consists of a set of query conditions; a query condition c is a tuple $(w(c), A(c), p(c), v(c))$ of a probabilistic weight $w(c) \in [0, 1]$ specifying the importance of this condition, a schema attribute $A(c) \in S$, a predicate $p(c)$ (supported by the data type $dt(A)$)) and a comparison value $v(c) \in dom(dt(A), p)$ Queries are normalised, i.e. the sum of the query condition weights $s(q) := \sum_{c \in q} w(c)$ equals 1:

$$q := \{(0.3, ti, =, xml), (0.2, ti, =, retrieval), (0.5, au, =, fuhr)\}.$$

As we consider different data types and predicates, we split the query q into subqueries q'_p w. r. t. the different predicates p. Then, we normalise q'_p and derive the subquery q_p (with the normalisation factor $s(q'_p)$). The query q_1 from above would be split into 2 subqueries:

$$q_{ti,=} = \{(0.6, ti, =, xml), (0.4, ti, =, retrieval)\}, q_{au,=} = \{(1, au, =, fuhr)\}.$$

We assume disjoint query terms, and view the query condition weight $w(c)$ as the probability $Pr(q \leftarrow c)$. Then, we can apply a linear retrieval function [19]:

$$Pr(q_p \leftarrow d) := \sum_{c_j \in q_p} \underbrace{Pr(q_p \leftarrow c_j)}_{\text{query condition weight}} \cdot \underbrace{Pr(c_j \leftarrow d).}_{\text{indexing weight}}$$

The probability $Pr(c \leftarrow d)$ is the indexing weight of document d w. r. t. condition c. Of course, the notion "indexing weight" does not imply that this probability is actually stored in an index, it can also be computed on the fly.

We are interested in the probability of relevance $Pr(\text{rel}|q_p, d)$, so we use predicate-specific mapping functions [13] for approximating the relationship between probabilities $Pr(\text{rel}|q_p, d)$ and $Pr(q_p \leftarrow d)$:

$$f_p : [0, 1] \mapsto [0, 1], \quad f_p(Pr(q_p \leftarrow d)) \approx Pr(\text{rel}|q_p, d). \tag{8}$$

We can convert the probabilities of relevance of the subqueries into a probability of relevance for the complete document:

$$Pr(\text{rel}|q, d) := \sum_{p \in q} s(q'_p) \cdot Pr(\text{rel}|q_p, d).$$

In the rest of this section, we present definitions for indexing weights $Pr(c \leftarrow d)$ and mapping functions f_p for different data types and predicates.

3.2 Data Type "Text"

For `Text`, the comparison value of a condition c is a term t. In MIND, we use two predicates `contains` (with stemming and stop word removal) and `containsNoStem` (terms are not stemmed, but stop words are removed). We use a modified BM25 scheme [16] for the indexing weights:

$$P(t \leftarrow d) := \frac{tf(t,d)}{tf(t,d) + 0.5 + 1.5 \cdot \frac{dl(d)}{avgdl}} \cdot \frac{\log \frac{numdl}{df(t)}}{\log |DL|}. \qquad (9)$$

Here, $tf(t,d)$ is the term frequency (number of times term t occurs in document d), $dl(d)$ denotes the document length (number of terms in document d), $avgdl$ the average document length, $numdl$ the sample or library size (number of documents), $|DL|$ the library size), and $df(t)$ the document frequency (number of documents containing term t). We modified the standard BM25 formula by the normalisation component $1/\log|DL|$ to ensure that indexing weights are always in the closed interval $[0,1]$, and can thus be regarded as a probability. The resulting indexing weights are rather small; but this can be compensated by the mapping functions.

Experiments [14] showed that a logistic function [4,5], defined by two parameters b_0 and b_1, yields a good approximation quality (see also Fig. 1):

$$f_p : [0,1] \to [0,1],\ f(x) := \frac{\exp(b_0 + b_1 x)}{1 + \exp(b_0 + b_1 x)}. \qquad (10)$$

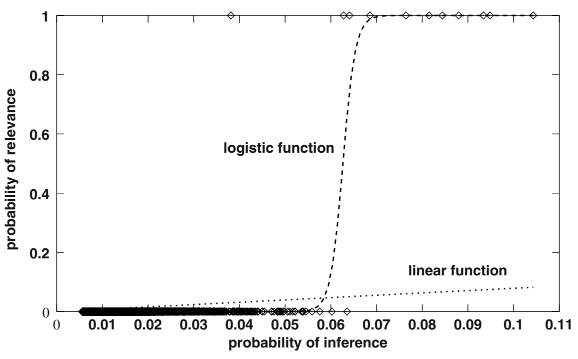


Fig. 1. Example query results and fit with linear and logistic function

The logistic function can also be justified by a theoretic point of view: In an ideal situation, exactly the documents in the ranks $1, \ldots, l$ are relevant, and the documents in the remaining ranks $l{+}1, \ldots$ are irrelevant. Let a be the probability of inference of the documents in rank l (i.e., the lowest probability of inference of any relevant documents). Then, the relationship function should be a step function

$$f(x) := \begin{cases} 1 & , \quad x \geq a, \\ 0 & , \quad x < a \end{cases} . \tag{11}$$

Obviously, no information retrieval system can ensure this requirement. Thus, in general some of the top-ranked documents are irrelevant, and some of the lower-ranked documents are relevant. But, less documents with lower probabilities of inference should be relevant than documents with higher probabilities. In other words, the probability that any arbitrary document is relevant should decrease with decreasing probability of inference.

For modelling this characteristics, we want a continuous function f which approximates the discrete step function (11). The logistic function in (10) is such an approximation.

The MIND consortium also investigated transcripts of speech recognisers, and observed that they can be handled in the same way as "ordinary" text.

3.3 Data Type "Name"

This data type supports two Boolean predicates `=` and `sounds-like`:

$$Pr(c \leftarrow d) \in \{0, 1\},$$

$$Pr((\mathit{author}, \mathit{sounds} - \mathit{like}, \mathit{Jones}) \leftarrow (\mathit{author}, \mathit{Johnson})) = 1.$$

For Boolean predicates, the identity function is a natural choice for the mapping function:

$$f_p \equiv id \ , \ f_p(x) := x. \tag{12}$$

Vague predicates like `edit-distance` or `n-grams` could be considered as well but are not investigated yet.

3.4 Data Type "Year"

The data type `Year` has the Boolean predicates `=`, `<`, `>`, `<=` and `>=`, i.e. the indexing weight is in $\{0, 1\}$. As for the data type `Name`, we use the identity mapping function.

Vagueness is required e.g. when a user is uncertain about the exact publication year of a document. E.g., if documents from the year 1999 are requested, a document from the year 2000 might also be relevant (although the probability is lower). Thus, we define three different vague predicates $\sim$`=`, $\sim$`<` and $\sim$`>`:

$$Pr(c_{\sim>} \leftarrow d) := \begin{cases} 1 - \frac{v(c) - v(d)}{v(d)} & , \quad v(c) > v(d) \\ 1 & , \quad \mathit{else} \end{cases} , \tag{13}$$

$$Pr(c_{\sim=} \leftarrow d) := 1 - \left(\frac{v(c) - v(d)}{v(d)} \right)^2 . \tag{14}$$

For example, we get

$$Pr((\mathit{pub}, \sim=, \mathit{1999}) \leftarrow (\mathit{pub}, \mathit{2000})) = 1 - \left(\frac{1999 - 2000}{2000} \right)^2 = 0.99999975.$$

For these vague predicates, we apply a logistic function.

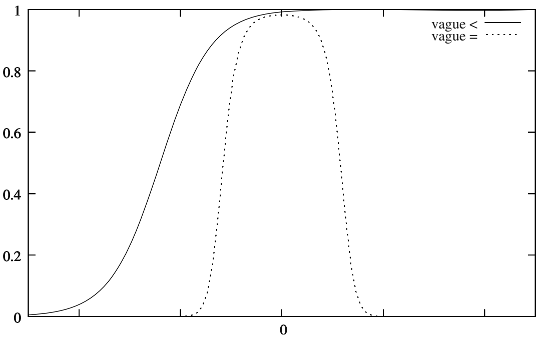


Fig. 2. Logistic indexing functions in fact retrieval

For ∼=, useful parameters are $b_0 = 4.7054 - 12590100$, $b_1 = 12590100$. Then, we get

$$b_0 + b_1 \cdot 0.99999975 \approx 1.5578, \quad \exp(b_0 + b_1 \cdot 0.99999975) \approx 4.7487,$$
$$f(0.99999975) \approx 0.8260.$$

These definitions of the indexing weights and mapping function are equivalent to the retrieval function used in fact retrieval for single condition queries (see Fig. 2), and a close approximation for queries with multiple conditions.

3.5 Data Type "Image"

For images, different similarity functions (usually distance measures) for comparing the condition value c and the image stored in document d can be considered. For colour histograms, the colour space is divided into bins. A histogram $hist(c)$ (comparison value in query) or $hist(d)$ (document image) is a vector H, where the component H_i counts the frequency of the colours in bin i in the image

Possible distance measures for colour histograms include Minkowski-form distance, Kullback-Leibler divergence, χ^2 statistics or a quadratic-form distance [17], [10], [11], or a normalised histogram intersection measure for the indexing weights:

$$Pr(c \leftarrow d) := \frac{\sum_i \min(hist(c)_i, hist(d)_i)}{\sum_i hist(d)_i}. \tag{15}$$

$$Pr(((image, colour, (0.5, 0.7)) \leftarrow (image, (0.4, 0.2))) = \frac{0.4 + 0.2}{0.7 + 0.3} = 0.6.$$

For images, linear or logistic mapping functions are possible.

3.6 Parameter Learning

Most mapping functions depend on some query-specifiy parameters. However, in practice we do not have query-specific relevance data, so we can only derive query-independent parameters from a learning sample (for each digital library).

In MIND, the parameters are learned using the Gnuplot[1] implementation of the nonlinear least-squares (NLLS) Marquardt-Levenberg algorithm [15]. As we don't have relevance judgements for all documents in practice, we only considered the 100 top-ranked documents.

4 Estimating Retrieval Quality for Resource Selection

Resource selection accuracy in this model heavily depends on good approximations of the number of relevant documents in the result set. Within MIND, we developed two new methods [13]: DTF-sample (simulated retrieval on sample) can be used for all data types, whereas DTF-normal (modelling indexing weights by a normal distribution) only works for text (but there, it outperforms DTF-sample). The third and oldest method DTF-rp [7] can only be used for linear mapping functions.

In this section we describe briefly these three methods for estimating the expected number $E[r(s,q)]$ of relevant documents in the first s documents. All of them require some metadata (e.g. average indexing weights, document frequencies). In non-co-operating environments, these "resource descriptions" can be created automatically by query-based sampling [1]. "Random" subsequent queries are submitted to the library, and the retrieved documents are collected (forming the "sample", from which the metadata is extracted). With reasonably low costs, an accurate resource description can be constructed from samples of e.g. 300 documents.

4.1 Recall-Precision-Function

With the linear retrieval function for $Pr(q \leftarrow d)$, a linear mapping functions $f(x) := c_0 + c_1 \cdot x$ and a query q where all conditions refer to the same predicate (which allows us to use the same mapping function), we can compute the expected number $E(\mathrm{rel}|q, DL_i)$ of relevant documents in DL_i as:

$$E(\mathrm{rel}|q,DL) = |DL| \cdot E(\mathrm{rel}|q,d),$$

$$E(\mathrm{rel}|q,d) = c_0 + c_1 \cdot \sum_{c_j \in q} Pr(q \leftarrow c_j) \cdot \underbrace{\left[\frac{1}{|DL|} \sum_{d \in DL} Pr(c_j \leftarrow d)\right]}_{\text{average indexing weight}} .$$

Obviously, this can be extended towards queries referring to multiple data types.

In addition, DTF-rp approximates the recall-precision function of a DL by a linearly decreasing function (see Fig. 3)

$$P : [0,1] \to [0,1],\ P(R) := P^0 \cdot (1 - R). \tag{16}$$

With expected precision $E[r(s,q)]/s$ and expected recall $E[r(s,q)]/E(\mathrm{rel}|q,DL)$, we can estimate the number of relevant documents when retrieving s documents [7]:

[1] `http://www.ucc.ie/gnuplot/gnuplot.html`

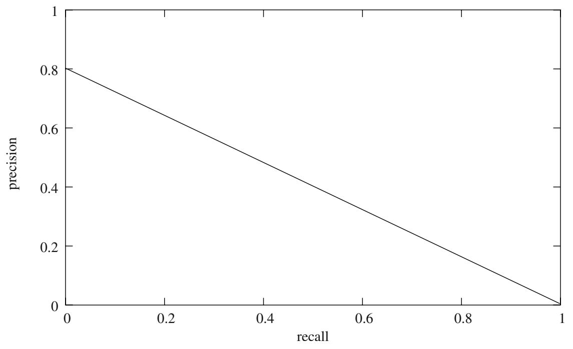


Fig. 3. Approximation of recall-precision function with $P^0 = P(0) = 0.8$

$$E[r(s,q)] := \frac{P^0 \cdot E(\mathrm{rel}|q, DL) \cdot s}{E(\mathrm{rel}|q, DL) + P^0 \cdot s}. \tag{17}$$

As this method only relies on the average indexing weights, it can be used for all data types.

4.2 Simulated Retrieval on Sample

For DTF-sample, we index the complete library sample instead of only extracting statistical metadata. In the resource selection phase, retrieval is simulated with the same query q on this small sample (e.g. 300 documents), obtaining a probability of relevance $Pr(\mathrm{rel}|q, d)$ for each sample document. This results in the empirical, discrete density p of the corresponding distribution.

Figure 4 shows how we can estimate the number of relevant documents in the result set of s documents. The grey area (the area below the graph from a to 1) denotes the fraction $s/|DL|$ of the documents in the library which are

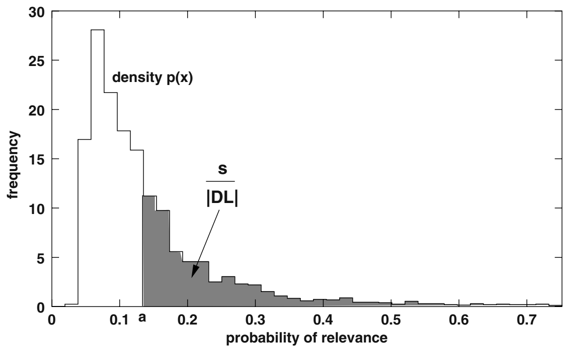


Fig. 4. Density of probabilities of relevance and computation of $E[r(s,q)]$

retrieved. Thus, we have to find a point $a \in [0,1]$ (the smallest probability of relevance among the s retrieved documents) such that

$$s = |DL| \int_a^1 p(x)\ dx. \tag{18}$$

With this point a, the expected number of relevant documents in the result set can be computed as the expectation of the probabilities of relevance in this area:

$$E[r(s,q)] = |DL| \int_a^1 p_i(x) \cdot x\ dx.$$

Obviously, this method can be used for all data types.

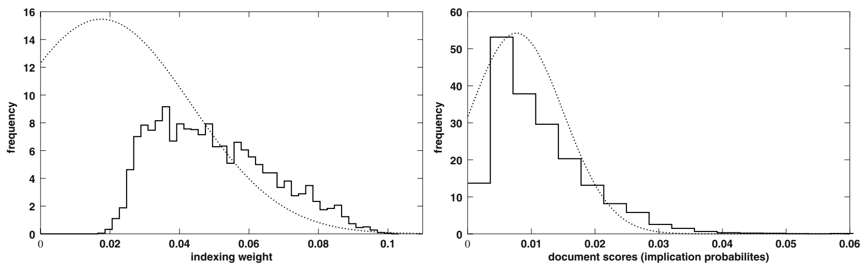
4.3 Normal Distribution

As DTF-sample, DTF-normal estimates the distribution of the probabilities of relevance $Pr(\mathrm{rel}|q,d)$, but based on a new theoretic model. For text, early experiments showed that the empirical, discrete distribution of the indexing weights $Pr(t \leftarrow d)$ (viewed as a random variable X_t for a term t) can be approximated by a normal distribution (defined by the expectation μ and the variance σ)

$$p(x,\mu,\sigma) := \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp(-\frac{(x-\mu)^2}{2\sigma^2}). \tag{19}$$

For improved readability, we left out a huge peak at zero in Fig. 5(a). This peak is a result of the large amount of documents which do not contain the term. The second effect is that the expectation is close to zero, and the normal distribution density is positive also for negative values. We ignore this, as we are mainly interested in the high indexing weights (the tail of the distribution).

To derive the distribution of probabilities of inference, we can view the indexing weights of term t in all documents in a library as a random variable X_t. Then, the distribution of the scores of all documents in a library is modelled by the random variable



(a) Indexing weights (b) Document scores

Fig. 5. Distributions with normal distribution fit

$$X := \sum_{i=1}^{l} a_i \cdot X_{t_i} \tag{20}$$

with the l query terms t_i and weights $a_i := Pr(q \leftarrow t_i)$. As X_{t_i} follow a normal distribution, the linear combination X is also distributed normally (see Fig. 5(b)), and the parameters can be computed efficiently:

$$\mu = \sum_{i=1}^{l} a_i \cdot \mu_{t_i}, \quad \sigma = \sqrt{\sum_{i=1}^{l} (a_i \cdot \sigma_{t_i})^2}. \tag{21}$$

From the document score distribution, we can estimate the scores (probabilities of inference) of the s top-ranked documents easily. By applying the mapping function, the probabilities of relevance can be computed.

This method can only be used for text, as the distribution of the indexing weights is unclear for other data types and predicates.

5 Evaluation

So far, we are only able to present an TREC-based evaluation for textual documents (the complete evaluation can be found in [13]). We compare DTF with CORI, the state-of-the-art representative of the current resource ranking systems.

For other data types like names, years or images, we do not have an appropriate test-bed with documents, queries and with relevance judgements.

5.1 Experimental Setup

We used the TREC-123 test bed with the CMU 100 library split [1]. The libraries are of roughly the same size (about 33 megabytes), but vary in the number of documents they contain. The documents inside a library are from the same source and the same time-frame. We took a constant number of 300 documents for each library. Table 1 depicts the summarised statistics for this 100 library test-bed.

We used the same document indexing terms and query terms for both CORI and the three DTF variants (using the Porter stemmer, and removing the TREC stop words). The document index only contains the `<text>` sections of the documents.

Queries are based on TREC topics 51–100 and 101–150 [9], respectively. We use three different sets of queries:

Short queries: only `<title>` field, between 1 and 7 terms (average 3.3), similar to those submitted to a web search engine.

Mid-length queries: only `<description>` field, between 4 and 19 terms (average 9.9), may be used by advanced searchers.

Long queries: all fields, between 39 and 185 terms (average 87.5), common in TREC-based evaluations.

Table 1. Summarised statistics for the 100 collection test-bed

(a) Complete libraries

	Minimum	Average	Maximum
Documents	752	10,782	33,723
Bytes	28,070,646	33,365,514	41,796,822

(b) Library samples

	Minimum	Average	Maximum
Documents	300	300	300
Bytes	229,915	2,701,449	15,917,750

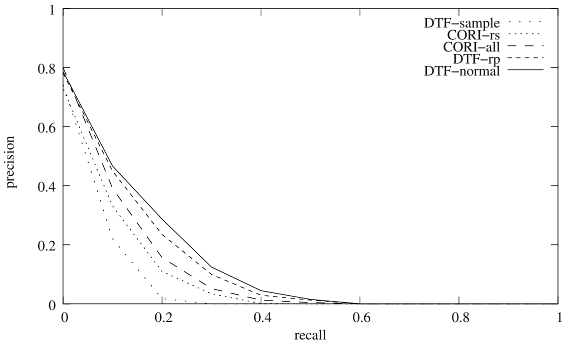


Fig. 6. Recall-precision graph for optimum experiments, long queries

The relevance judgements are the standard TREC relevance judgements [9], documents with no judgement are treated as irrelevant.

The standard weighting schemes for documents and queries are used for the CORI experiments. For the DTF experiments, the modified BM25 weighting scheme as described in (9) is employed. Normalised tf values are used as query term weights:

$$P(q \leftarrow t) := \frac{tf(t,q)}{ql(q)} \quad . \tag{22}$$

Here, $tf(t,q)$ denotes the term frequency (number of times a term t occurs in query q), and $ql(q) := \sum_{t \in q} \; tf(t,q)$ is the query length (number of terms in query q).

For the DTF experiments we used the same indexing and retrieval methods for the 100 libraries as we use for the resource selection index. We always requested 300 documents.

For CORI, resource selection, retrieval on the selected libraries and merging the results are all performed by CORI. For each of the 100 libraries we built another index using the INQUERY retrieval engine [2]. The 10 top-ranked libraries are selected, and 30 documents are retrieved from each (resulting in 300

Table 2. Precision in top ranks and average precision

(a) Short queries

	CORI	DTF-rp	DTF-sample	DTF-normal
5	0.4260 / +0.0%	0.4060 / -4.7%	0.3680 / -13.6%	0.3980 / -6.6%
10	0.3930 / +0.0%	0.3950 / +0.5%	0.3520 / -10.4%	0.3840 / -2.3%
15	0.3840 / +0.0%	0.3840 / +0.0%	0.3300 / -14.1%	0.3813 / -0.7%
20	0.3640 / +0.0%	0.3730 / +2.5%	0.3150 / -13.5%	0.3750 / +3.0%
30	0.3487 / +0.0%	0.3480 / -0.2%	0.2853 / -18.2%	0.3493 / +0.2%
Avg.	0.0517 / +0.0%	0.0662 / +28.0%	0.0349 / -32.5%	0.0645 / +24.8%

(b) Mid queries

	CORI	DTF-rp	DTF-sample	DTF-normal
5	0.3840 / +0.0%	0.4440 / +15.6%	0.3840 / +0.0%	0.4480 / +16.7%
10	0.3630 / +0.0%	0.4190 / +15.4%	0.3580 / -1.4%	0.4220 / +16.3%
15	0.3500 / +0.0%	0.3987 / +13.9%	0.3327 / -4.9%	0.4087 / +16.8%
20	0.3350 / +0.0%	0.3830 / +14.3%	0.3160 / -5.7%	0.3980 / +18.8%
30	0.3107 / +0.0%	0.3627 / +16.7%	0.2900 / -6.7%	0.3747 / +20.6%
Avg.	0.0437 / +0.0%	0.0605 / +38.4%	0.0297 / -32.0%	0.0703 / +60.9%

(c) Long queries

	CORI	DTF-rp	DTF-sample	DTF-normal
5	0.5780 / +0.0%	0.5880 / +1.7%	0.5200 / -10.0%	0.5780 / +0.0%
10	0.5590 / +0.0%	0.5680 / +1.6%	0.5130 / -8.2%	0.5690 / +1.8%
15	0.5340 / +0.0%	0.5440 / +1.9%	0.4927 / -7.7%	0.5587 / +4.6%
20	0.5175 / +0.0%	0.5400 / +4.3%	0.4605 / -11.0%	0.5450 / +5.3%
30	0.5013 / +0.0%	0.5233 / +4.4%	0.4260 / -15.0%	0.5323 / +6.2%
Avg.	0.0883 / +0.0%	0.1111 / +25.8%	0.0506 / -42.7%	0.1230 / +39.3%

documents all together as for DTF). These experiments are run with the Lemur toolkit implementation of CORI.[2]

DTF-rp, DTF-sample and DTF-normal all require a learning phase (see Sect. 3.6). The parameters are learned using a cross-evaluation strategy: Parameters are learned on TREC topics 51–100 and evaluated on topics 101–150, and vice versa.

5.2 Experimental Results

For optimum retrieval quality the DTF cost function only contains costs for retrieval quality ($C^+ = 0$ and $C^- = 1$), and the algorithms computes an optimum solution with a variable number of selected DLs (in contrast to CORI which always selects a fixed number of libraries). We learned DTF parameters

[2] `http://www-2.cs.cmu.edu/~lemur/`

separately for short, mid-length and long queries, and applied them on the same query type.

The results are depicted in Tab. 2. For all query types (short, mid-length and long queries), DTF-rp and DTF-normal perform about the same (DTF-sample performs bad all the time). Compared to CORI, the quality is better for mid-length and long queries (for short queries, only in some ranks). The improvement is maximal for mid-length queries, and competitive for short queries typically for the Web. For long queries (well-known from former CORI evaluations [6]), the improvement is better than for short queries but worse than for mid-length queries.

The recall-precision graph for long queries is depicted in Fig. 6; the graphs for the mid-length and the long queries are about the same and were left out due to space limitations. For long queries one can see that DTF-rp performs best, followed (in this order) by DTF-normal, CORI and DTF-sample.

6 Conclusion and Outlook

In this paper we introduce a resource selection method which can—in addition to text—also deal with other data types like names, years, images and others. This method is based on the decision-theoretic framework which assigns costs to document retrieval, and aims at minimising the overall costs from all selected libraries. In contrast to traditional resource ranking algorithms like GlOSS or CORI, DTF computes a clear cutoff for the number of libraries queried, and the number of documents which should be retrieved from each of these libraries. For DTF, the selection solution is a vector specifying for each library the number of documents which have to be retrieved.

In the decision-theoretic framework, data types play their role in estimating retrieval quality (time spent on a library and monetary costs are independent of the data types used in queries). Earlier work focused on text. However, two of the three methods for estimating retrieval quality—DTF-rp (recall-precision function) and DTF-sample (simulated retrieval on sample)— can also be used for non-text datatypes. The third method—DTF-normal (normal distribution of indexing weights)—can only be used for texts.

In future, we will extend our mapping function evaluations on other data types. The biggest challenge will be to find data sets using non-text data types together with relevance judgements (the evaluation for text is based on TREC data).

Acknowledgements. This work is supported by the EU commission under grant IST-2000-26061 (MIND).

References

[1] J. Callan and M. Connell. Query-based sampling of text databases. *ACM Transactions on Information Systems*, 19(2):97–130, 2001.

[2] J. Callan, W. Croft, and S. Harding. The INQUERY retrieval system. In *Proceedings of DEXA-92, 3rd International Conference on Database and Expert Systems Applications*, pages 78–83, Heidelberg et el., 1992. Springer.

[3] J. Callan, Z. Lu, and W. Croft. Searching distributed collections with inference networks. In E. Fox, P. Ingwersen, and R. Fidel, editors, *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 21–29, New York, 1995. ACM. ISBN 0-89791-714-6.

[4] S. Fienberg. *The Analysis of Cross-Classified Categorical Data*. MIT Press, Cambridge, Mass., 2. edition, 1980.

[5] D. Freeman. *Applied Categorical Data Analysis*. Dekker, New York, 1987.

[6] J. French, A. Powell, J. Callan, C. Viles, T. Emmitt, K. Prey, and Y. Mou. Comparing the performance of database selection algorithms. In *Proceedings of the 22nd International Conference on Research and Development in Information Retrieval*, pages 238–245, New York, 1999. ACM.

[7] N. Fuhr. A decision-theoretic approach to database selection in networked IR. *ACM Transactions on Information Systems*, 17(3):229–249, 1999.

[8] L. Gravano and H. Garcia-Molina. Generalizing GIOSS to vector-space databases and broker hierarchies. In U. Dayal, P. Gray, and S. Nishio, editors, *VLDB'95, Proceedings of 21th International Conference on Very Large Data Bases*, pages 78–89, Los Altos, California, 1995. Morgan Kaufman.

[9] D. Harman, editor. *The Second Text REtrieval Conference (TREC-2)*, Gaithersburg, Md. 20899, 1994. National Institute of Standards and Technology.

[10] S. Kullback. Information theory and statistics. Dover, New York, NY, 1968.

[11] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Petkovic, P. Yanker, C. Faloutsos, and G. Taubin. The QBIC project: Querying images by content using color, texture, and shape. volume 1908, pages 173–181. SPIE, 1993.

[12] H. Nottelmann and N. Fuhr. Combining DAML+OIL, XSLT and probabilistic logics for uncertain schema mappings in MIND. In *European Conference on Digital Libraries (ECDL 2003)*, Heidelberg et el., 2003. Springer.

[13] H. Nottelmann and N. Fuhr. Evaluating different methods of estimating retrieval quality for resource selection. In *Proceedings of the 26st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, New York, 2003. ACM.

[14] H. Nottelmann and N. Fuhr. From uncertain inference to probability of relevance for advanced IR applications. In F. Sebastiani, editor, *25th European Conference on Information Retrieval Research (ECIR 2003)*, pages 235–250, Heidelberg et el., 2003. Springer.

[15] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, editors. *Nested Relations and Complex Objects in Databases*. Cambridge University Press, 1992.

[16] S. E. Robertson, S. Walker, M. Hancock-Beaulieu, A. Gull, and M. Lau. Okapi at TREC. In *Text REtrieval Conference*, pages 21–30, 1992.

[17] M. Swain and D. Ballard. Color indexing. *International Journal of Computer Vision*, 7(1):11–32, 1991.

[18] C. J. van Rijsbergen. A non-classical logic for information retrieval. *The Computer Journal*, 29(6):481–485, 1986.

[19] S. Wong and Y. Yao. On modeling information retrieval with probabilistic inference. *ACM Transactions on Information Systems*, 13(1):38–68, 1995.

Distributed Web Search as a Stochastic Game[⋆]

Rinat Khoussainov and Nicholas Kushmerick

Department of Computer Science
University College Dublin
Belfield, Dublin 4, Ireland
{rinat,nick}@ucd.ie

Abstract. Distributed search systems are an emerging phenomenon in Web search, in which independent topic-specific search engines provide search services, and metasearchers distribute user's queries to only the most suitable search engines. Previous research has investigated methods for engine selection and merging of search results (i.e. performance improvements from the user's perspective). We focus instead on performance from the service provider's point of view (e.g, income from queries processed vs. resources used to answer them). We analyse a scenario in which individual search engines compete for user queries by choosing which documents (topics) to index. The challenge is that the utilities of an engine's actions should depend on the uncertain actions of competitors. Thus, naive strategies (e.g, blindly indexing lots of popular documents) are ineffective. We model the competition between search engines as a stochastic game, and propose a reinforcement learning approach to managing search index contents. We evaluate our approach using a large log of user queries to 47 real search engines.

1 Introduction

Distributed heterogeneous search environments are an emerging phenomenon in Web search. Consider a federation of *independently controlled* metasearchers and many specialised search engines. The specialised search engines provide focused search services in a specific domain (e.g. a particular topic). Metasearchers help to process user queries effectively and efficiently by distributing them only to the most suitable search engines for each query. Compared to the traditional search engines like Google, specialised search engines (together) may provide access to arguably much larger volumes of high-quality information resources, frequently called "deep" or "invisible" Web.

We envisage that such heterogeneous environments will become more popular and influential. However, to unlock the benefits of distributed search for the users, there must be an incentive for search engines to participate in such a federation, i.e. an opportunity to make money. Previous research has mainly

⋆ This research was supported by grant SFI/01/F.1/C015 from Science Foundation Ireland, and grant N00014-03-1-0274 from the US Office of Naval Research.

targeted performance of such environments from the user's perspective (i.e. improved quality of search results). On the other hand, a provider of search services is more interested in the utility of the service compared to the cost of the resources used (e.g. the income from search queries processed versus the amount of resources needed for answering those queries).

An important factor that affects a particular search engine's performance in heterogeneous search environments is *competition* with other independently controlled search engines. When there are many engines available, users will send queries to those that would provide the best possible results. Thus, the service offered by one search engine influences queries received by its competitors. Multiple search providers can be viewed as participants in a search services market competing for user queries. We examine the problem of performance-maximising behaviour for non-cooperative specialised search engines in heterogeneous search environments. In particular, we analyse a scenario in which independent topic-specific search engines compete for user queries by choosing which documents (topics) to index.

Example. *Consider a heterogeneous environment with two search engines A and B having equal resource capabilities. Assume that users are only interested in either "sport" or "cooking", with "sport" being more popular. If A and B each decide to index both "sport" and "cooking" (i.e. everything, like Google tries to do), they will receive an equal share of all user queries. If A decides to spend all its resources only on "sport" while B stays on both topics, A will provide better search for "sport" than B. Then users will send queries on "sport" to A, and on "cooking" to B. Therefore, A will receive more queries (and so will have higher performance). If, however, B also decides to index only "sport", both search engines will compete only for the same queries and, thus, will each receive even fewer requests than in the 2 previous cases.*

While the search engines in a heterogeneous search environment are independent in terms of selecting their content, they are not independent in terms of the performance achieved. Actions of one search engine affect the queries received by its competitors, and vice versa. The uncertainty about competitors as well as the potentially large number of competing engines make our optimisation problem difficult. For example, in one experiment (Sec. 4), several search engines that competed head-to-head for the most popular topic were less profitable than an engine that cornered the market on another topic that was less popular but ignored by the competition. This example illustrates that naive strategies such as blindly indexing popular documents can be suboptimal.

We model the competition between specialised search engines as a partially observable stochastic game, and exploit the concept of *"bounded rationality"* [1]. Bounded rationality assumes that decision makers are unable to act optimally in the game-theoretic sense due to incomplete information about the environment and/or limited computational resources. We cast our problem as a reinforcement learning task, where the goal of a specialised search engine is to learn a good behaviour strategy against given (potentially sub-optimal) competitors. The effectiveness of our approach is evaluated in a simulation environment. The simulator implements a simplified formalisation of the problem and is driven by user queries submitted to over 47 existing search engines.

2 Problem Formalisation

The questions of optimal behaviour in computational markets have been researched extensively from both consumers and suppliers points of view [2,3]. However, services markets in heterogeneous Web search environments have a number of features requiring new techniques. Given the complexity of the problem domain, attempting to make our analysis 100% realistic from the very beginning is neither feasible nor reasonable. Instead, we start with a simplified model. Our goal is to select an approach that allows us in principle to factor more realistic details into our models in future.

2.1 Performance Metric

We adopt an economic view on search engine performance. Performance is a difference between the value (utility) of the search service provided and the cost of the resources used to provide the service. The value of a search service is a function of the user queries processed. The cost structure in an actual search engine may be quite complicated involving many categories, such as storage, crawling, indexing, and searching. We use the following formula for the search engine performance: $P = \alpha_1 Q - \alpha_2 QD - \alpha_3 C - \alpha_4 D$, where Q is the number of queries received in a given time interval, D is the number of documents in the engine's index, C is the number of new documents added to the index during the given time interval, and α_x are constants.

$\alpha_1 Q$ represents the service value: if the price of processing one search request for a user is α_1, then $\alpha_1 Q$ would be the total income from service provisioning. $\alpha_2 QD$ represents the cost of processing search requests. If x amount of resources is sufficient to process Q queries, then we need $2x$ to process twice as many queries in the same time. Similarly, twice as many resources are needed to search twice as many documents in the same time. Thus, the amount of resources can be expressed as $\alpha_2 QD$, where α_2 reflects the resource costs. The example of the FAST search engine confirms that our cost function is not that far from reality [4]. $\alpha_3 C$ is the cost of crawling and indexing new documents. While the cost of indexing new documents is indeed proportional to the number of the documents added, the cost of crawling can vary between documents. Here, we rather assume an average crawling cost. Finally, $\alpha_4 D$ is the cost of document storage and maintenance. This includes storing a copy of the document as well as keeping the document description up-to-date.

We assume that all search engines use the same α_x constants when calculating their performance. Having the same α_2–α_4 reasonably assumes that the cost of resources (CPU, memory, network) per "unit" is the same for all search engines. Having the same α_1 assumes, perhaps unrealistically, that the search engines choose to charge users the same amount per query. We leave to future work the optimisation problem in environments where engines may have different service pricing.

2.2 Engine Selection Model

We use a very generic model of the metasearch component, so we can abstract from implementation details or particular metasearch algorithms. Essentially, we assume that users would like to send queries to the search engine(s) that contain the most relevant documents to the query, and the more of them, the better. The ultimate goal of the metasearcher is to select for each user query search engines that maximise the results relevance, while minimising the number of engines involved. The existing research in metasearch (e.g. [5]), however, does not go much further than simply ranking search engines. We assume that the query is always forwarded to the *highest ranked* search engine. In case several search engines have the same top rank, one is selected at random.

The ranking of search engines is based on the expected number of relevant documents that are indexed by each engine. The engine i that indexes the largest expected number of documents NR_i^q relevant to query q will have the highest rank. We apply a probabilistic information retrieval approach to assessing relevance of documents [6]. For each document d, there is a probability $\Pr(rel|q,d)$ that this document will be considered by the user as relevant to query q. In this case, $NR_i^q = \sum_{d \in i} \Pr(rel|q,d)$, where $d \in i$ iterates over the documents indexed by engine i. If $\Pr(rel|q_1,d) = \Pr(rel|q_2,d), \forall d$ then queries q_1 and q_2 will look the same from both the metasearcher's and search engine's points of view, even though $q_1 \neq q_2$. Therefore, all queries can be partitioned into equivalence classes with identical $\Pr(rel|q,d)$ functions. We call such classes *topics*. We assume that there is a fixed finite set of topics and queries can be assigned to topics. One way to approximate topics in practice would be to cluster user queries received in the past and then assign new queries to the nearest clusters.

2.3 Engine Selection for "Ideal" Crawlers

Assume that users only issue queries on a single topic. We will see later how this can be extended to multiple topics. To receive queries, a search engine needs to be the highest ranked one for this topic. Given an index size D, engine i would like to index a set of D documents with the largest possible NR_i (expected number of documents relevant to the topic).

Search engines use topic-specific (focused) Web crawlers to find documents to index. Since it is very difficult to model a Web crawler, we assume that all search engines have "ideal" Web crawlers which for a given D can find the D most relevant documents on a given topic. Under this assumption, two search engines indexing the same number of documents $D_1 = D_2$ will have $NR_1 = NR_2$. Similarly, if $D_1 < D_2$, then $NR_1 < NR_2$ (if all documents have $\Pr(rel|d) > 0$). Therefore, the metasearcher will forward user queries to the engine(s) containing the largest number of documents.

This model can be extended to multiple topics, if we assume that each document can only be relevant to a single topic. In this case, the state of a search engine can be represented by the number of documents D_i^t that engine i indexes for each topic t. A query on topic t will be forwarded to the engine i with the largest D_i^t.

2.4 Decision Making Sequence

The decision making process proceeds in series of fixed-length time intervals. For each time interval, search engines simultaneously and independently decide on how many documents to index on each topic. They also allocate the appropriate resources according to their expectations for the number of queries that users will submit during the interval (incurring the corresponding costs). Since engines cannot have unlimited crawling resources, we presume that they can only do incremental adjustments to their index contents that require the same time for all engines. The users submit queries during the time interval, which are allocated to the search engines based on their index parameters (D_i^t) as described above. Then the process repeats.

Let $\hat{Q}_i^t$ be the number of queries on topic t that, according to expectations of search engine i, the users will submit. Then the total number of queries expected by engine i can be calculated as $\hat{Q}_i = \sum_{t:D_i^t>0} \hat{Q}_i^t$. Obviously, we only expect queries for those topics, for which we index documents (i.e. for which $D_i^t > 0$). We assume that engines always allocate resources for the full amount of queries expected, so that in case they win the competition, they will be able to answer all queries received. Then the cost of resources allocated by engine i can be expressed as $\alpha_2 \hat{Q}_i D_i - \alpha_3 C_i - \alpha_4 D_i$, where $D_i = \sum_t D_i^t$ is the total number of documents indexed by engine i, and $C_i = \sum_t C_i^t$ is the total number of documents added to the engine's index in this time interval. For the given resource allocation, $\hat{Q}_i$ will be the total number of queries that engine i can process within the time interval (its query processing capacity).

The number of queries on topic t *actually forwarded* to engine i (presuming that $D_i^t > 0$) can be represented as $Q_i^t = 0$ if engine i is ranked lower than its competitors (i.e. $\exists j, D_i^t < D_j^t$), and $Q_i^t = Q^t/|K|$ if i is in the set of the highest-ranked engines $K = \{k : D_k^t = \max_j D_j^t\}$. Q^t is the number of queries on topic t *actually submitted* by the users. The total number of queries forwarded to search engine i can be calculated as $Q_i = \sum_{t:D_i^t>0} Q_i^t$. We assume that if the search engine receives more queries than it expected (i.e. more queries than it can process), the excess queries are simply rejected. Therefore, the total number of queries processed by search engine i equals to $\min(Q_i, \hat{Q}_i)$.

Finally, performance of engine i over a given time interval can be represented as follows: $P_i = \alpha_1 \min(Q_i, \hat{Q}_i) - \alpha_2 \hat{Q}_i D_i - \alpha_3 C_i - \alpha_4 D_i$. Note that even if an engine wins in the competition, its performance decreases as the index grows, and eventually becomes negative. This effect accords with the intuition that a huge index must eventually cost more to maintain than can ever be recovered by answering queries, and serves to justify our economic framework for analysing optimal search engine behaviour.

2.5 Decision Making as a Stochastic Game

The decision making process can be modelled as a stochastic game. A *stochastic game* (SG) [7] is a tuple $\langle n, S, A_{1...n}, T, R_{1...n} \rangle$, where n is the number of decision makers (players), S is a set of game states, A_i is a set of actions for player i, $T : S \times A \times S \to [0, 1]$ is a transition function (where $A = A_1 \times ... \times A_n$), and $R_i : S \times A \to \mathbb{R}$ is a reward function for player i. SGs are very similar to Markov Decision Processes (MDPs) except there are multiple decision makers and the next state and rewards depend on the joint action of the players.

In our case, players are search engines, the state of the game at stage k is defined by the state of the indices of all search engines $((D_1^t(k)), ..., (D_n^t(k)))$ and by time (which determines user queries at the given stage). A player's action $a_i(k) \in A_i$ is a vector $(a_i^t(k))$ of index adjustments for each topic t. The following index adjustments a_i^t are possible: *Grow* (increase the number of documents indexed on topic t by one); *Same* (do not change the number of documents on topic t); and *Shrink* (decrease the number of documents on topic t by one). The reward to player i is calculated using the formula for P_i, where $C_i^t = 1$ if $a_i^t(k)$ in this time interval was "Grow", and $C_i^t = 0$ otherwise.

2.6 Strategies and Observations

A player's strategy (policy) in the game is a function that maps a history of the player's current (and possibly past) observations of the game to a probability distribution over player's actions. In our SG, a player's observations consist of two parts: observations of the state of its own search engine and, observations of the opponents' state. For T topics, the player's inputs consist of T observations of the state of its own search engine (one for each topic) and T observations of the relative positions of the opponents (one per topic), i.e. $2T$ observations in total.

The observations of its own state reflect the number of documents in the search engine's index for each topic t. We do not assume that search engines know the contents of each other's indices. Instead, observations of the opponents' state reflect the relative position of the opponents in the metasearcher rankings, which indirectly gives the player information about the states of the opponents' index. The following three observations are available for each topic t: *Winning* – there are opponents ranked higher for topic t than our search engine (i.e. they index more documents on the topic than we do); *Tying* – there are opponents having the same and smaller ranks for topic t than our search engine (opponents index the same and smaller number of documents on the topic); and *Losing* – the rank of our search engine for topic t is higher than opponents (opponents index less documents on the topic than we do).

How can a player know relative rankings of its opponents? It can send a query on the topic of interest to the metasearcher (as a search user) and request a ranked list of search engines for the query. We also assume that players can obtain from the metasearcher information (statistics) on the queries previously submitted by user. This data are used in calculation of the expected number of queries for each topic $\hat{Q}_i^t$. In particular, the number of queries on topic t expected by engine i at stage k equals to the number of queries on topic t submitted by users at the previous stage (i.e. $\hat{Q}_i^t(k) = Q^t(k-1)$).

3 The COUGAR Approach

Optimal behaviour in an SG is in general opponent-dependent: to select their future actions, players need to know future opponents' strategies. Nash equilibrium from game theory [8,7] provides a way to resolve this uncertainty about opponents: if players agree to play a Nash equilibrium, then they do not have incentive to unilaterally deviate (thus they become certain about each other's future actions).

Agreeing to play a Nash equilibrium, however, is problematic in games with multiple equilibria (which is frequently the case). There is a large body of research on equilibrium selection in game theory. Ultimately, it requires characterising all Nash equilibria of a game, which is NP-hard even given complete information about the game [9]. These results and the possibility that players may not have complete information lead to the idea of "bounded rationality", when players are limited in their abilities to make optimal decisions. Our goal is to learn a strategy that performs well against the given opponents rather than trying to calculate some equilibrium behaviour.

Learning in SGs have been studied extensively in game theory and machine learning. We propose to use a recent reinforcement learning algorithm called GAPS (which stands for **G**radient **A**scent for **P**olicy **S**earch) [10]. In GAPS, the learner plays a parameterised strategy represented by a non-deterministic Moore automaton, where the parameters are the probabilities of actions and state transitions. GAPS implements stochastic gradient ascent in the space of policy parameters. After each learning trial, parameters of the policy are updated by following the reward gradient.

GAPS has a number of advantages important for our problem domain. Unlike model-based reinforcement learning algorithms, GAPS does not attempt to build a model of the game or the opponents from the interaction experience, and thus can cope with partial observability and scale well with the number of opponents. The policies learned by GAPS can be both non-deterministic and state-full. The ability to play non-deterministic policies means that GAPS can potentially achieve the optimal performance in the games where only mixed (non-deterministic) strategy equilibria exist [8]. Learning state-full policies can be advantageous in partially observable settings [11]. Finally, GAPS scales well to multiple topics by modelling decision-making as a game with factored actions (where action components correspond to topics). The action space in such games is the product of factor spaces for each action component. GAPS, however, allows us to reduce the learning complexity: rather than learning in the product action space, separate GAPS learners can be used for each action component. It has been shown that such distributed learning is equivalent to learning in the product action space [10]. As with most gradient-based methods, the disadvantage of GAPS is that it is only guaranteed to find a local optimum.

We call a search engine that uses the proposed approach COUGAR (**CO**mpetitor **U**sing **G**APS **A**gainst **R**ivals). The COUGAR controllers are represented by a set of non-deterministic Moore automata (M^t), one for each topic, functioning synchronously. Inputs of each automaton M^t are game observations for topic t. Outputs of each automaton are actions changing the number of documents D_i^t indexed for topic t. The resulting action of the controller is the product of actions (one for each topic) produced by each of the individual automata.

Training of COUGAR is performed in series of learning trials. Each learning *trial* consists of 100 days, where each day corresponds to one stage of the SG played. The search engines start with empty indices and then, driven by their controllers, adjust their index contents. In the beginning of each day, engine controllers receive observations and simultaneously produce control actions (change their document indices). Users issue a stream of search queries for one day. The metasearcher distributes queries between the engines according to their index parameters on the day, and the search engines collect the corresponding rewards. For the next day, the search engines continue from their current states, and users issue the next batch of queries. The resulting performance in the whole trial is calculated as a sum of discounted rewards from each day. After each trial, the COUGAR controller updates its strategy using the GAPS algorithm. That is, the action and state transition probabilities of the controller's automata are modified using the payoff gradient (see [12] for details).

To simulate user search queries, we used HTTP logs obtained from a Web proxy of a large ISP and extracted queries to 47 well-known search engines. The total number of queries extracted was 657,861 collected over a period of 190 days, and we used a 100-day subset of these queries in our simulations. We associated topics with search terms in the logs. To simulate queries for T topics, we extracted the T most popular terms from the logs.

4 Experimental Results

Our experiments consist of two parts. In the first part, we evaluate the performance of COUGAR against various fixed opponents. In the second part, we test COUGAR against evolving opponents, which in our case are also COUGAR learners. In the experiments with fixed opponents, we simulated 2 search engines competing for 2 topics.

4.1 The "Bubble" Opponent

The "Bubble" strategy tries to index as many documents as possible without any regard to what competitors are doing. From our performance formula, such unconstrained growing leads eventually to negative performance. Once the total

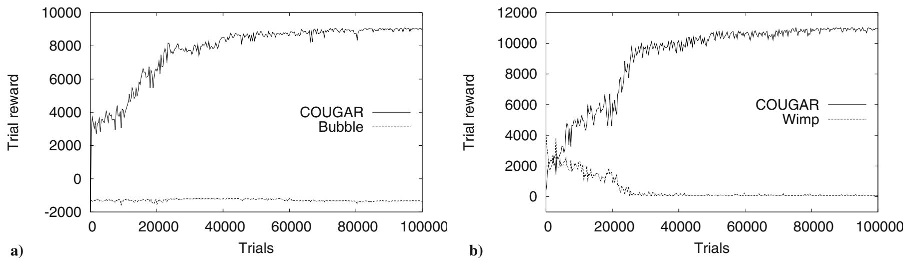


Fig. 1. Learning curves: (a) COUGAR vs “Bubble”; (b) COUGAR vs “Wimp”.

reward falls below a certain threshold, the “Bubble” search engine goes bankrupt (it shrinks its index to 0 documents). This process imitates the situation in which a search provider expands its business without paying attention to costs, eventually runs out of money, and quits. An intuitively sensible response to the “Bubble” strategy would be to wait until the bubble “bursts” and then come into the game alone. That is, a competitor should not index anything while the “Bubble” grows and should start indexing a minimal number of documents once “Bubble” goes bankrupt.

Fig. 1a shows how COUGAR’s performance improved during learning. Once the COUGAR engine reached a steady performance level, its resulting strategy was evaluated in a series of testing trials. Analysis of a sample testing trial shows that COUGAR has learned to wait until “Bubble” goes bankrupt, and then to win all queries for both topics.

4.2 The “Wimp” Opponent

The “Wimp” controller used a more intelligent strategy. Consider it first for the case of a single topic. The set of all possible document index sizes is divided by “Wimp” into three non-overlapping sequential regions: “Confident”, “Unsure”, and “Panic”. The “Wimp’s” behaviour in each region is as follows: *Confident* – the strategy in this region is to increase the document index size until it ranks higher than the opponent. Once this goal is achieved, ”Wimp” stops growing and keeps the index unchanged; *Unsure* – in this region, “Wimp” keeps the index unchanged, if it is ranked higher or the same as the opponent. Otherwise, it retires (i.e. reduces the index size to 0); *Panic* – “Wimp” retires straight away.

An overall idea is that “Wimp” tries to outperform its opponent while in the “Confident” region by growing the index. When the index grows into the “Unsure” region, “Wimp” prefers retirement to competition, unless it is already winning over or tying with the opponent. This reflects the fact that the potential losses in the “Unsure” region (if the opponent wins) become substantial, so “Wimp” does not dare to risk.

To generalise the “Wimp” strategy to multiple topics, it was modified in the following way. When assessing its own index size, “Wimp” was simply adding

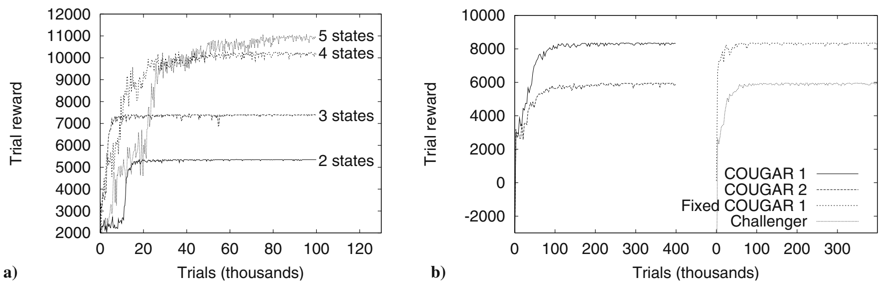


Fig. 2. (a) "Wimp" vs COUGAR with different number of states; (b) COUGAR in self play: learning (left), evaluation (right).

the documents for different topics together. Similarly, when observing relative positions of the opponent, it was adding together ranking scores for different topics. Finally, like the multi-topic "Bubble", "Wimp" was changing its index size synchronously for each topic. Common sense tells us that one should behave aggressively against "Wimp" in the beginning (i.e. index more than "Wimp"), to knock him out of competition, and then enjoy the benefits of monopoly (i.e. index minimum possible number of documents). This is what COUGAR has learned to do. Fig. 1b also presents the learning curve.

4.3 Policy Complexity and Performance

As pointed out earlier, the ability of COUGAR to learn non-deterministic and state-full policies can be a potential advantage under partial observability. A policy with more states can represent more complex (and potentially more optimal) behaviours. To analyse the effects of the policy complexity on the performance achieved, we evaluated COUGAR controllers with different number of states against the "Wimp" opponent. Fig. 2a demonstrates that indeed COUGARs with more states performed better, though in most cases learned more slowly.

4.4 Performance Bounds

While COUGAR was superior to both fixed strategy opponents, an interesting question is how its performance relates to the maximum and minimum values obtainable for the given opponents. To asses this, we evaluated "Bubble" and "Wimp" against the corresponding best-case (omniscient) and worst-case strategies. COUGAR has achieved 99.8% of the optimal performance against "Bubble" and 88.6% against "Wimp". These facts not only show COUGAR's near-optimality but also demonstrate the fact that such performance was not a result of dumb opponents: COUGAR could have performed much worse.

4.5 COUGAR in Self Play

It is not guaranteed from the theoretical point of view that the gradient-based learning will always converge in self play. In practice, however, we observed that learners converged to relatively stable strategies. The simulation setup was the same as for the fixed opponent experiments. We trained two COUGAR learners, each using a 3-state GAPS policy, in self play in a scenario with two topics. The players decided to split the query market: each search engine specialised on a different topic. The winner happened to pick the more popular topic.

As previously mentioned, a policy learned by COUGAR in self-play is, in general, only locally optimal against the policies simultaneously learned by its opponents. A question we are interested in is how well such locally optimal policy can perform against an unbiased COUGAR learner. To test this, we fixed the strategy of the winner from self play, and trained another COUGAR against it. We can envisage a possible failure of the winner's policy, if it did not learn to compete well for the more popular topic (because of its opponent), and learned not to compete for the less popular topic. Then an unbiased challenger might be able to capture queries for both topics. Evaluation against a 3-state COUGAR challenger showed, however, that the performance of the winner's policy is quite stable. Fig. 2b shows the performance of the search engines during self play (left) and evaluation (right). The fixed COUGAR achieves the same performance against the challenger as it had in self play.

4.6 Scaling up

An important advantage of GAPS is the ability to handle well multiple topics by modelling decision-making as a game with factored actions. To analyse the scalability of our approach with the number of topics, we simulated 10 COUGAR learners competing in a scenario with 10 different topics. Analysis showed that, similarly to the case of 2 learners and 2 topics, they decided to split the query market. An important detail is that more popular topics attracted more search engines. This confirms that the market split was really profit-driven and demonstrates COUGAR's "economic thinking". However, the most profitable engines actually specialised on the relatively unpopular topics, while the engines that competed for the most popular topics ended up earning substantially less money. Therefore, a naive strategy of indexing the most popular documents was suboptimal in this experiment.

5 Conclusions

Stochastic games provide a convenient theoretical framework for modelling competition between search engines in heterogeneous Web search systems. However, finding optimal behaviour strategies in stochastic games is a challenging task, especially in partially observable games, and also when other players may evolve over time. We demonstrated that reinforcement learning with state-full policies seems an effective approach to the problem of managing the search engine content. Our adaptive search engine, COUGAR, has shown the potential to derive behaviour strategies allowing it to win in the competition against non-trivial fixed opponents, while policies learned in self-play were robust enough against evolving opponents (other COUGARs in our case).

We do not claim to provide a complete solution for the problem here, but we believe it is the promising first step. Clearly, we have made many strong assumptions in our models. One future direction will be to relax these assumptions to make our simulations more realistic. In particular, we intend to perform experiments with real documents and using some existing metasearch algorithm. We currently assume that users are insensitive to the quality of the search results; we intend to extend our metasearch and service pricing models to account for user satisfaction. Another direction would be to further investigate the issues of the learning convergence and performance robustness in self-play, and against other learners.

References

1. Rubinstein, A.: Modelling Bounded Rationality. The MIT Press (1997)
2. Greenwald, A., Kephart, J., Tesauro, G.: Strategic pricebot dynamics. In: Proc. of the 1st ACM Conf. on Electronic Commerce. (1999) 58–67
3. J.Hu, Wellman, M.: Learning about other agents in a dynamic multiagent system. Cognitive Systems Research **2** (2001)
4. Risvik, K., Michelsen, R.: Search engines and Web dynamics. Computer Networks **39** (2002)
5. Gravano, L., Garcia-Molina, H.: GlOSS: Text-source discovery over the Internet. ACM Trans. on Database Systems **24** (1999) 229–264
6. van Rijsbergen, C.J.: Information Retrieval. 2nd edn. Butterworths (1979)
7. Filar, J., Vrieze, K.: Competitive Markov Decision Processes. Springer Verlag, New York (1997)
8. Osborne, M., Rubinstein, A.: A Course in Game Theory. The MIT Press (1999)
9. Conitzer, V., Sandholm, T.: Complexity results about Nash equilibria. In: Proc. of the 18th Intl. Joint Conf. on AI. (2003)
10. Peshkin, L., Meuleau, N., Kim, K.E., L.Kaelbling: Learning to cooperate via policy search. In: Proc. of the 16th Conf. on Uncertainty in AI. (2000)
11. Singh, S., Jaakkola, T., Jordan, M.: Learning without state-estimation in partially observable Markovian decision processes. In: Proc. of the 11th Intl. Conf. on Machine Learning. (1994)
12. Peshkin, L.: Reinforcement learning by policy search. MIT AI Lab Technical Report 2003-003 (2002)

Collection Fusion for Distributed Image Retrieval

S. Berretti, A. Del Bimbo, and P. Pala*

Dipartimento Sistemi e Informatica
Università di Firenze
via S.Marta 3, 50139 Firenze, Italy
{berretti,delbimbo,pala}@dsi.unifi.it

Abstract. Searching information through the Internet often requires users to contact several digital libraries, author a query representing the information of interest and manually gather retrieved results. However, a user may be not aware of the content of each individual library in terms of quantity, quality, information type, provenance and likely relevance, thus making effective retrieval quite difficult.
Searching distributed information in a network of libraries can be simplified by using a centralized server that acts as a gateway between the user and distributed repositories. To efficiently accomplish this task, the centralized server should perform some major operations, such as *resource selection*, *query transformation* and *data fusion*. Resource selection is required to forward the user query only to the repositories that are candidate to contain relevant documents. Query transformation is necessary in order to translate the query into one or more formats such that each library can process the query. Finally, data fusion is used to gather all retrieved documents and conveniently arrange them for presentation to the user.
In this paper, we introduce an original framework for collection fusion in the context of image databases. In fact, the continuous nature of content descriptors used to describe image content, makes impractical the applicability of methods developed for text. The proposed approach splits the score normalization process into a learning phase, taking place off-line, and a normalization phase that rearranges scores of retrieved images at query time, using information collected during the learning. Fusion examples and results on the accuracy of the solution are reported.

1 Introduction

Nowadays, many different document repositories are accessible through the Internet. A generic user looking for a particular information should contact each repository and verify the presence of the information of interest. This typically takes place by contacting the repository server through a client web browsing

* This work is supported by EU Commission under grant IST-2000-26061. Detailed information about this project is available at http://www.mind-project.org

application. Then, the user is presented a web page that guides her/him in the process of retrieval of information (in a multimedia scenario this can be in the form of text documents, images, videos, audio and so on).

This framework has two major drawbacks. First of all, it is assumed that the user is aware of the presence of all potentially relevant repositories, in terms of quantity, quality, information type, provenance and likely relevance. Internet search engines such as *Altavista*, *Google* and *Lycos*, to say a few, can provide only limited support to accomplish this task. A second drawback is related to the fact that, in order to find the information of interest, the user typically contacts several repositories. Each one is queried and even for a limited number of repositories, the user is soon overwhelmed by a huge and unmanageable amount of (probably irrelevant) retrieved documents.

Federated digital libraries have been recently proposed as a solution to these problems. A possible architecture for a federation of digital libraries is characterized by the presence of a central server acting as a gateway between the user and all federated resources. In particular, a generic user looking for some information sends the query to the server. This one is in charge to forward the query to all (or a subset of all) the networked resources. All retrieved documents (that is the set of all documents returned by each resource) are gathered by the central server and conveniently arranged for presentation to the user.

Implementation of this solution requires the availability of some modules that take care of summarizing the content of a resource (*resource description*), use resource summaries to identify which resources contain documents relevant to a query (*resource selection*), translate the query in the specific format accepted by each resource (*query transformation*), issue the query to the resources identified as relevant, merge retrieval results in decreasing order of relevance so that they can be presented to the user (*collection fusion*).

Resource description, resource selection and collection fusion for distributed libraries have been widely investigated in the past focussing on libraries of text documents [2], [3], [4], [5], [8], [9], [10].

However, solutions developed so far for libraries of text documents exploit the peculiar characteristics of content descriptions of textual materials. Typically, these descriptions are in the form of term and document frequencies. The former represent, for each vocabulary term, the percentage of documents that contain at least one occurrence of the term. The latter represent the occurrence of a term within one document. Since the vocabulary is composed of a finite number of elements, variables describing the content of textual materials take values over discrete sets.

Differently, content descriptors of multimedia documents—such as audio files, images, videos, Flash presentations—are in the form of feature vectors taking dense values over continuous spaces. This complicates (if not makes it impossible) the applicability of solutions developed for text to the domain of multimedia libraries.

In this paper, we present a novel method to accomplish collection fusion for distributed libraries of images. Although the proposed method is presented in the context of image libraries, its domain of applicability includes generic multimedia libraries. Indeed, the proposed solution can be applied in all those contexts where document content is in the form of feature vectors, representing instance values of metric attributes.

In general, the fusion process is intended as a mean to overcome difficulties related to the assessment of the relative relevance of retrieved items. These difficulties are particularly challenging if retrieved documents are not associated with matching scores—quite a common case for text retrieval engines but not as common for image retrieval engines. However, the availability of matching scores for all retrieved documents doesn't solve the problem completely. In fact, matching scores are not comparable across different collections since different collections may use retrieval engines with different similarity metrics and/or different content descriptors.

The proposed solution relies on a model based approach where, for each library, a normalization model is defined to map scores assigned by the library into normalized scores. Once documents returned by all libraries have been associated with their normalized scores, the relevance of each document with respect to each other can be assessed. The normalization model is identified by a set of parameters (the model parameters) that determine the function mapping original scores into their normalized counterparts. Use of the normalization model is articulated in two distinct phases: *learning* and *normalization*. During learning, each library is processed through a set of *sampling queries* so as to learn values of model parameters for that particular library. Once values of model parameters have been learned for all federated libraries, normalization can be accomplished. Normalization takes place during a query session and allows, given the user query and results returned by all libraries, original scores assigned to retrieved documents to be transformed into normalized scores.

This paper is organized as follows. In Sect.2, we briefly review previous work on data fusion. Sect.3 addresses the proposed solution, introducing the learning and normalization phases for data fusion. Experimental results are reported in Sect.4. Finally, Sect.5 gives conclusions and future directions of work.

2 Previous Work

Several merging strategies have been proposed to deal with merging results returned by distributed libraries (mostly of text documents). In general, a common distinction used by researchers in this area is the difference between data fusion and collection fusion [11]. The former takes place in a setting where all of the retrieval systems involved have access to the same text collection. The latter is used when the collections searched by all retrieval systems are disjoint.

One of the most known approaches is called *Round-Robin* [10]. In this approach, it is assumed that each collection contains approximately the same number of relevant items that are equally distributed on the top of the result lists provided by each collection. Therefore, results merging can be accomplished by picking up items from the top of the result lists in a round-robin fashion (the first item from the first list, the first from the second list, ..., the second from the first list and so on).

Unfortunately, the assumption of uniform distribution of relevant retrieved items is rarely observed in practice, especially if libraries are generic collections available on the Web. This drastically reduces the effectiveness of the round-robin approach.

A different solution develops on the hypotheses that *i*) for each retrieved document, its matching score is available and *ii*) the same search model is used to retrieve items from different collections. In this case, document matching scores are comparable across different collections and they can be used to drive the fusion strategy. This approach is known as *Raw Score Merging* [6], but it is rarely used due to the large number of different search models that are applied to index documents even in text libraries.

To overcome limitations of approaches based on raw score merging or Round Robin, more effective solutions have been proposed developing on the idea of score normalization. These solutions assume that each library returns a list of documents with matching scores. In this case, some technique is used to normalize matching scores provided by different libraries. For instance, score normalization can be accomplished by looking for duplicate documents in different lists. The presence of one document in two different lists is exploited to normalize scores in the two lists. Unfortunately, this approach cannot be used if one or more lists do not contain documents retrieved also by other retrieval engines.

A more sophisticated approach, based on cross similarity of documents, is presented in [7]. Retrieved documents are used to build a local collection. Then, the similarity of each retrieved document to each document in the local collection is evaluated. Cross similarities between documents are used to normalize matching scores. The main limitations of this approach are related to its computational complexity (cross similarities between all pairs of retrieved documents should be computed) and to the fact that it requires the extraction of content descriptors from each retrieved document. This latter requirement can be easily accomplished for text documents, but implies critical computational requirements if applied to images.

In order to lessen these requirements, in [9] a new approach is presented that is based on the use of a local archive of documents to accomplish score normalization. Periodically, libraries are sampled in order to extract a set of representative documents. Representative documents extracted from all the libraries are gathered into a central data fusion archive. When a new query is issued to the libraries, it is also issued to the central search engine that evaluates the similarity of the query with all documents in the central data fusion archive. Then, under the assumption that each retrieved list contains at least two documents that are also included in the central data fusion archive, linear regression is exploited to compute normalization coefficient (for each retrieved list) and normalize scores. This approach has been successfully tested for text libraries for which resource descriptions were extracted using *Query based Sampling* [2].

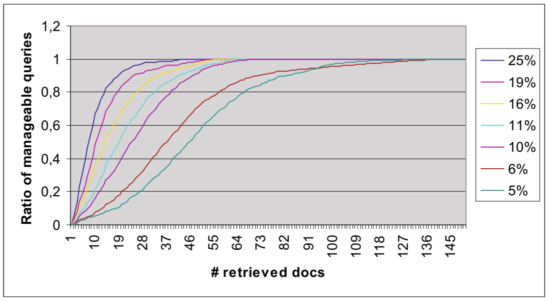


Fig. 1. Applicability (on an average of 1000 random queries) of fusion based on sampling and regression

In order to appreciate pro's and con's of this approach, it is necessary to describe briefly how query based sampling works and how it can affect the fusion process. Query based sampling is a method to extract a description of the content of a resource by iteratively running queries to the resource and analyzing retrieved documents. For each retrieved document, two main actions are performed: *i*) a description of document content is extracted and stored locally; *ii*) some document elements (such as words and phrases) are used to formulate new queries to the resource. At the end of the iterative process, content descriptors of retrieved documents are used to compute a description of the whole resource content.

If query based sampling is used in combination with the approach presented in [9] to accomplish collection fusion, each document retrieved during the query based sampling session is also stored in the central data fusion archive.

Thanks to this framework, when a user query is issued to the system, resource descriptions of all federated libraries are processed for the purpose of selecting those libraries that are best candidate to contain relevant documents. Basically, a library is selected if elements of its resource description match the query. If this happens, it means that during the query based sampling session many documents were retrieved from the library—and stored in the central data fusion archive—with a content similar to the query. Hence, this means that very likely, a large number of retrieved documents will also be present in the central data fusion archive.

However, if resource descriptions are not built using query based sampling, the assumption that retrieval results of each library contain at least two documents of the central archive is rarely verified. In these cases, linear regression cannot be applied and matching scores are left un-normalized. Thus, results cannot be merged.

As an example, in Fig.1 some plots are shown that evidence the applicability of this approach to a library of images not described using query based sampling. In particular plots report (on an average of 1000 random queries) the occurrence of cases where the approach is applicable (y-axis) with respect to the number of retrieved documents (x-axis). Several plots are shown corresponding to resource descriptions with different sizes. As expected, the larger the number of retrieved documents, the higher the applicability of the approach—the larger the number of retrieved documents the lower the probability that there are not at least two documents that are also included in the resource description. For the same reason, the larger the resource description size, the higher the probability of success. However, if the number of retrieved documents is small the probability of success becomes unacceptably low. For instance, if only 30 documents are retrieved and the resource description size is 11% of the library size, the probability of success is only 80%.

3 Data Fusion for Image Libraries

The proposed solution develops on the method presented in [9]. Improvements are introduced to address two major limitations of the original method: applicability of the solution to the general case, regardless of the particular solution adopted for resource description extraction; reduction of computational costs at retrieval time. These goals are achieved by decomposing the fusion process in two separate steps: *model learning* and *normalization*.

Model learning is carried out once, separately for each library: it is based on running a set of *sampling queries* to the library and processing retrieval results. In particular, for each sampling query, the library returns a list of retrieved images with associated matching scores. These images are reordered using a fusion search engine that associates with each image a normalized score. Then, the function mapping original scores onto normalized scores is learned.

Differently, normalization takes place only at retrieval time. It allows, given a query and results provided by a library for that specific query, to normalize matching scores associated with retrieved results. This is achieved by using the normalization model learned for that library during the model learning phase.

In doing so, collection fusion is achieved through a model learning approach: during the first phase parameters of the model are learned; during the second one the learned model is used to accomplish score normalization and enable fusion of results based on normalized scores merging. Model learning and normalization are described in detail in the following sections.

3.1 Score Modelling

Let $\mathcal{L}^{(i)}(q) = \{(d_k, s_k)\}_{k=1}^{n}$ be the set of pairs document/score retrieved by the i-th digital library as a result to query q. We assume that the relationship between un-normalized scores s_k and their normalized counterpart σ_k can be approximated to a linear relationship. In particular, we assume $\sigma_k = a * s_k + b + \epsilon_k$, being ϵ_k an approximation error (*residue*), a and b two parameters that depend on the digital library (i.e. they may not be the same for two different digital libraries, even if the query is the same) and on the query (i.e. for one digital library they may not be the same for two different queries). The approximation error ϵ_k accounts for the use of heterogeneous content descriptors (the digital library and the data fusion server may represent image content in different ways) as well as for the use of different similarity measures (even if the digital library and the data fusion server use the same image content descriptors—e.g. 64 bins color histogram—they may use two different similarity measures—e.g. one may compare histograms through a quadratic form distance function, the other may use the histogram intersection).

Values of parameters a and b are unknown at the beginning of the learning process. However, during learning, values of these parameters are computed, separately for each library. For each sample query, the value of parameters a and b is estimated. Computing the value of parameters a and b for different queries is equivalent to sampling the value distribution of these parameters on a grid of points in the query space. Knowledge of parameter values on the grid points is exploited to approximate the value of parameters for new queries. This is accomplished by considering the position of the new query with respect to grid points.

Approximation of parameters a and b for a new query is carried out as follows. If the new query matches exactly one of the sample queries (say q_k) used during the learning process, then the value of a and b is set exactly to the value of a and b that was computed for the sample query q_k. Otherwise, the three grid points q_j, q_k, q_l that are closest to the new query are considered. The value of a and b for the new query is approximated by considering values of a and b as they were computed for queries q_j, q_k, q_l. In particular, if these values were a_j, b_j, a_k, b_k, a_l and b_l, then values of a and b are estimated as:

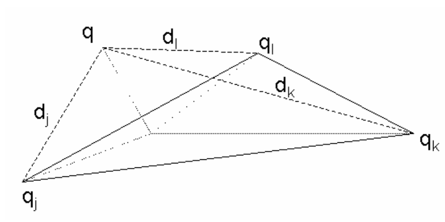
$$\begin{aligned} a &= \frac{d_j}{D}a_j + \frac{d_k}{D}a_k + \frac{d_l}{D}a_l \\ b &= \frac{d_j}{D}b_j + \frac{d_k}{D}b_k + \frac{d_l}{D}b_l \end{aligned} \tag{1}$$

being $D = d_j + d_k + d_l$ and d_j, d_k and d_l the Euclidean distance between the new query and grid points q_j, q_k, q_l, respectively (see Fig.2(a)).

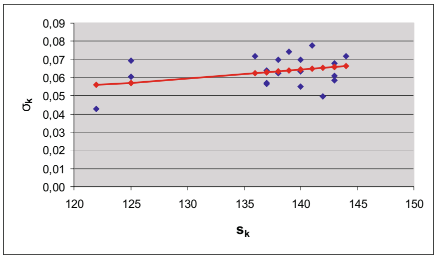
Using Eqs.1 is equivalent to estimate values of parameters a and b as if the new query lain to the hyperplane passing by q_j, q_k and q_l. For this assumption to be valid, the new query should be close to the hyperplane. The closer it is, the more precise the approximation.

In the proposed approach, linear regression is the mathematical tool used for the estimation of normalization coefficients, given a sample query and the corresponding retrieved documents. In our case, data points are pairs $(\sigma_i,\ s_i)$ being s_i the original matching score of the i-th document and σ_i its normalized score. The application of this method to a collection of pairs $(\sigma_i,\ s_i)$ results in the identification of the two parameters a and b such that $\sigma_i = a * s_i + b + e_i$.

Fig.2(b) plots, for a representative sample query, the values of normalized scores σ_k with respect to un-normalized ones s_k. The straight line is that derived during the learning phase by computing parameters a and b of the linear



(a)



(b)

Fig. 2. (a) Position of a new query wrt the approximating points on the grid; (b) Normalized scores σ_k with respect to un-normalized ones s_k for a sample query, and the straight line approximating their relationship.

regression between scores σ_k and s_k. This shows that the linear model assumption is quite well fulfilled, at least for the first few retrieved images.

3.2 Selection of Sample Queries

Through the learning phase. distribution of parameters a and b is sampled for a set of queries. Selection of these *sample queries* affects the quality of the approximation of a and b for a generic query in the query space during normalization.

Basically, two main approaches can be distinguished to select sample queries. The first approach completely disregards any information available about the distribution of documents in the library. In this case, the only viable solution is to use sample queries uniformly distributed in the query space. A different approach relies on the availability of information about the distribution of documents in the library. This should not be considered a severe limitation since this kind of information is certainly available to accomplish the resource selection task. In particular, this information is available in the form of a *resource descriptor* capturing the content of the entire library. Typically, in the case of image collections, resource descriptors are obtained by clustering library documents and retaining information about cluster centers, cluster population and cluster radius. Clustering is performed at several resolution levels so as to obtain multiple cluster sets, each one representing the content of the library at a specific *granularity* [1].

Resource descriptors can be used to guide the selection of sample queries by using the cluster center themselves as sample queries. In this way, the distribution of sample queries conforms to the distribution of documents in the library.

Once the resource description process is completed, each cluster center c_k is used as a query to feed the library search engine. This returns a set of retrieved images and associated matching scores s_i. Then, c_k is used as a query to feed the reference search engine (i.e. the search engine used by the data fuser) that associates with each retrieved image a normalized (reference) matching score σ_i. Linear regression is used to compute regression coefficients (m_k, q_k) for the set of pairs (σ_i, s_i). In doing so, each cluster center c_k is associated with a pair (a_k, b_k) approximating values of regression coefficients for query points close to c_k. The set of points (c_k, a_k, b_k) can be used to approximate the value of regression coefficients (a, b) for a generic query q by interpolating values of (m_k, q_k) according to Eqs.1.

4 Experimental Results

The proposed solution to data fusion for distributed digital libraries has been implemented and tested on a benchmark of three different libraries each one including about 1000 images. Two libraries represent image content through 64 bins color histograms. However, they use two different similarity measures, namely L_1 (*city-block*) and L_2 (*Euclidean*) norm, in order to compare image histograms. The third library computes local histograms on the components of a 3×3 grid which partitions the image. The CIE *Lab* is used as reference color space for the computation of this histogram.

As an example, Fig.3 shows results of the fusion process for two of the libraries and a sample query.

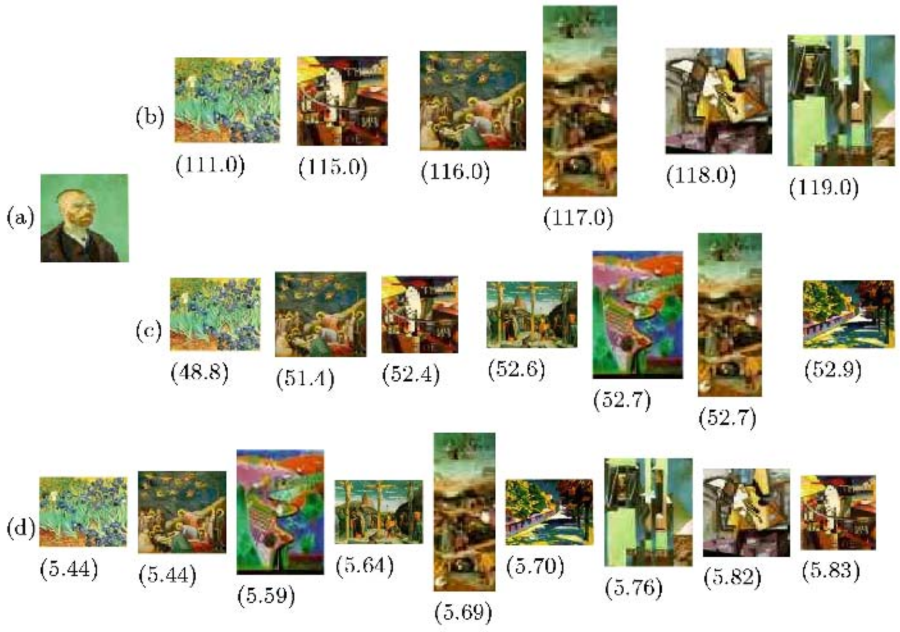


Fig. 3. Fusion of results for two distinct collections. (a) The query image; (b) images and matching scores retrieved from the first library; (c) images and matching scores retrieved from the second library; (d) fused results with normalized matching scores.

The query image is shown in Fig.3(a). Results returned by the first library and their original scores are shown in Fig.3(b). The same information is shown in Fig.3(c) for the second library. Fused results, ordered based on their normalized scores are shown in Fig.3(d). It can be noticed that, based on the un-normalized scores returned by the two libraries, all the images in Fig.3(c) would be ranked before images in (b), with a potential loss of relevant results if, for example, only the first eight images are returned to the user. Differently, application of the proposed data fusion technique allows a more effective re-ranking of retrieved images lists, which better conforms to the user expectation.

Experimental results are presented to report both on the quality of normalization coefficients approximation and on the overall quality of the data fusion process.

Each library was processed separately. Libraries were first subject to the extraction of resource descriptors according to the procedure presented in [1]. Cluster centers extracted through the resource description process were used as sample queries. For each sample query, values of normalization coefficients were computed, as explained in Sect.3.1.

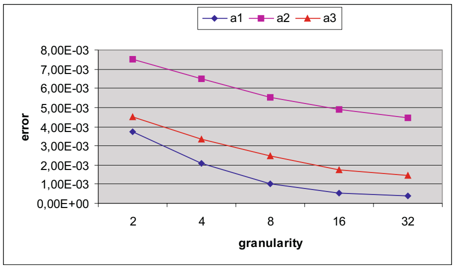
Evaluation of the proposed solution is carried out addressing two different accuracy measures. The first one describes the accuracy associated with approximation of normalization coefficients. The second one describes the accuracy associated with ranking of merged results.

Values of normalization coefficients for sample queries were used to approximate the value of normalization coefficients for new queries. In particular, the accuracy of the approximation was measured with reference to a random set of test queries, not used during model learning. For each test query, values of normalization coefficients were approximated according to Eqs.(1). Actual values of normalization coefficients were also computed by running the query and following the same procedure used for model learning (Sect.3.1). Comparison of actual and approximated values of normalization coefficients gives a measure of the approximation accuracy.

Let a and b be the estimated value of normalization parameters and $\hat{a}$ and $\hat{b}$ their actual values. Quality of the approximation is measured by considering the expected value of the normalized errors:

$$\epsilon_a = E\left[\frac{|a-\hat{a}|}{1+|\hat{a}|}\right], \quad \epsilon_b = E\left[\frac{|b-\hat{b}|}{1+|\hat{b}|}\right] \tag{2}$$

Plots in Fig.4 report values of the expected normalized errors for different values of the granularity of resource descriptions. Granularity is measured as the percentage of resource descriptions cardinality with respect to the database size. According to this, a smaller granularity corresponds to a lower number of sample documents used at learning time. It can be noticed that, even with coarse descriptors corresponding to low granularity levels, the value of the approximation error is always quite small. This experimentally confirms that it is possible to use estimated parameters for the data fusion process instead of computing their actual values on the fly at retrieval time.



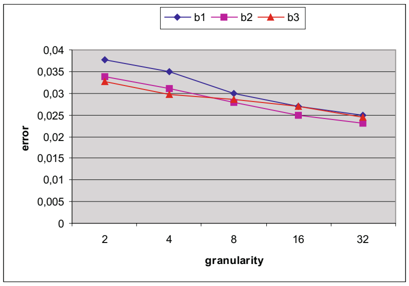


Fig. 4. Normalization error for approximated parameters of the three digital libraries: (a) error on parameter a; (b) error on parameter b.

The second measure of accuracy aims to estimate the quality of the data fusion and of the ordering of retrieved images that it yields. This is obtained by considering how images are ordered using an image search engine to process all retrieved images. This results in the comparison of two ordered lists of images.

The first list (*data fusion* list) is the output of the proposed data fusion module (called *Learning and Regression*, *LR* in the following). The fusion module receives the N best ranked retrieved images from each library and transforms their original scores into normalized scores. This yields a list of $N * L$ documents (being L the number of available digital libraries) ordered by normalized scores.

A second ranked list (*reference* list) is obtained by ranking the set of all $N * L$ retrieved documents using the local image search engine. To build this second list, each one of the $N * L$ images is processed in order to extract its content descriptor. Then, these content descriptors are compared against the user query and all images are sorted in decreasing order of similarity. For this experimentation, the content of each image is described in terms of a color histogram which is different from those used by the digital libraries. The similarity between two image histograms is evaluated through the *histogram intersection* distance.

Let $\mathcal{L}^{(r)}(q) = \{(d_k^{(r)}, \sigma_k^{(r)})\}_{k=1}^{M}$ and $\mathcal{L}^{(df)}(q) = \{(d_k^{(df)}, \sigma_k^{(df)})\}_{k=1}^{M}$ be the set of pairs document/normalized-score for the reference and the data fusion list, respectively, obtained by combining results returned from a query q. The size of both the lists is indicated as $M = N * L$. Let $f : \{1, \ldots, M\} \mapsto \{1, \ldots, M\}$ a function that associates the index i of a generic document in the reference list to the index $f(i)$ of the same document in the data fusion list. We assume that the best result for the data fusion process corresponds to the case in which every document in the data fusion list has the same rank that the document yields in the reference list, that is: $rank^{(r)}(i) = rank^{(df)}(f(i))$. Thus, a measure of the overall quality of the data fusion process can be expressed through the following functional:

$$\mathcal{F} = \frac{1}{N} \sum_{i=1,N} |rank^{(r)}(i) - rank^{(df)}(f(i))| \tag{3}$$

that is, the average value of rank disparity computed on the first N images of the two lists. The lower the value of $\mathcal{F}$ the higher the quality of data fusion (i.e. 0 corresponds to an optimal data fusion).

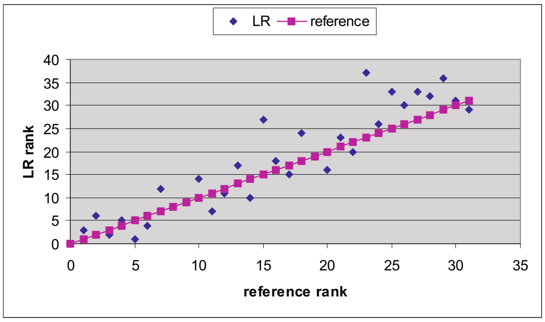


Fig. 5. Quality of the data fusion for a sample query. The straight diagonal line is the ideal mapping which preserves the image rankings between the reference and the data fuser list.

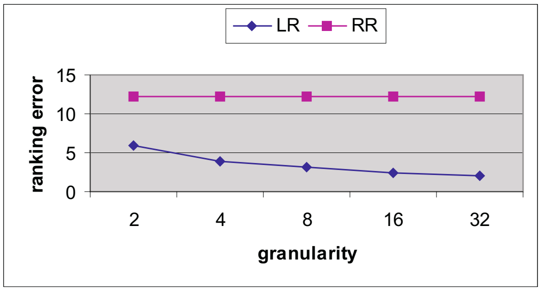


Fig. 6. Quality of the data fusion for the Learning and Regression method (LR curve) and for the Round-Robin approach (RR curve). Values of $\mathcal{F}$ are reported for different granularities.

As an example, Fig.5 plots values of $\mathcal{F}(i) = |rank^{(r)}(i) - rank^{(df)}(f(i))|$ for a sample query and resource descriptions granularity equal to 4. In this experiments the three libraries are considered ($L = 3$), each returning sixteen images ($N = 16$), so that the maximum size of the fused and reference lists is equal to $M = 16 \times 3 = 48$. Resulting values are reported only for the first 32 images ranked according to the reference list. The horizontal axis represents the ranking produced according to the reference list, while the vertical axis is the ranking for the list originated by the data fuser. In this graphic, the straight diagonal line represents the behavior of an ideal fuser, which yields the same ranking given by the reference list. Each displacement with respect to this line indicates a difference in ranking the corresponding document between the two lists.

It can be observed that the error is quite small for all ranks. In particular, the error is small for the first and more relevant images of the ranking. In addition only few of the images comprised in the first 32 of the reference list are ranked outside of this limit in the LR list. A similar behavior has been experimented in the average case.

Finally, the proposed LR solution for data fusion has been compared to the general Round-Robin (RR) approach [10].

Fig.6 compares values of $\mathcal{F}$ for LR and RR, averaged on 100 random queries and for different granularities. It results that the average ranking error for LR is quite small and decreases with the granularity. It can be also observed that the Learning and Regression solution yields better results with respect to RR approach, with the gain of LR increasing with the granularity.

5 Conclusion and Future Work

In this paper, an approach is presented for merging results returned by distributed image libraries. The proposed solution is based on learning and approximating normalization coefficients for each library. Since the major computational effort is moved to the learning phase, the approach has a low processing time at retrieval time, making it applicable to archives of images. In addition, differently from other fusion methods which rely on the presence of common documents in different result lists, the proposed fusion strategy is completely general, not imposing any particular constraint on the results. Experimental results are reported to demonstrate the effectiveness of the proposed solution. A comparative evaluation has also shown that the proposed Linear and Regression solution outperforms the general Round-Robin approach to the data fusion problem.

Future work, will address experimentation and comparison of the effect of the learning technique on fusion effectiveness. Moreover, issues related to the relationships between granularity of sample queries and accuracy of fusion will be investigated.

Acknowledgements. This work is supported by the European Community under IST Project MIND (IST-2000-26061).

References

1. S. Berretti, A. Del Bimbo, P. Pala. Using Indexing Structures for Resource Descriptors Extraction from Distributed Image Repositories. *In Proc. IEEE Int. Conf. on Multimedia and Expo*, vol.2, pp. 197–200, Lousanne, Switzerland, August 2002.
2. J. Callan and M. Connell. Query-based sampling of text databases. *ACM Transactions on Information Systems*, vol.19, n.2, pp. 97–130, 2001.

3. W. Chang, G. Sheikholaslami, A. Zhang, T. Syeda-Mahmood. Efficient Resource Selection in Distributed Visual Information Retrieval. In *Proc. of ACM Multimedia'97*, Seattle, 1997.
4. N. Fuhr. Optimum Database Selection in Networked IR. *In Proc. of the SIGIR'96 Workshop on Networked Information Retrieval*, Zurich, Switzerland, August 1996.
5. L. Gavarno and H. Garcia-Molina. Generalizing Gloss to Vector-Space Databases and Broker Hierarchies. *In Proc. of the 21st Int. Conf. on Very Large Data Bases*, pp. 78–89, 1995.
6. K.L. Kwok, L. Grunfeld, D.D. Lewis. TREC-3 Ad-hoc, Routing Retrieval and Thresholding Experiment using PIRCS. *In Proc. of TREC-3*, 1995, pp. 247–255.
7. S.T. Kirsch. Document Retrieval Over Networks wherein Ranking and Relevance Scores are Computed at the Client for Multiple Database Documents. US patent 5659732.
8. H.V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K.C. Sevcik, T. Suel. Optimal Histograms with Quality Guarantees". VLDB 1998, pp. 275–286.
9. L. Si and J. Callan. Using sampled data and regression to merge search engine results. *In Proc. of International ACM SIGIR Conference on Research and Development in Information Retrieval* pp. 19–26, Tampere, Finland, 2002.
10. E.M. Vorhees, N.K. Gupta, B. Johnson-Laird. Learning Collection Fusion Strategies. *In Proc. ACM-SIGIR'95*, 1995, pp.172-179.
11. E.M. Vorhees, N.K. Gupta, B. Johnson-Laird. The Collection Fusion Problem. *In The Third Text REtrieval Conference (TREC-3).*

New Methods of Results Merging for Distributed Information Retrieval

Shengli Wu, Fabio Crestani, and Forbes Gibb

Department of Computer and Information Sciences
University of Strathclyde, Glasgow, Scotland, UK
{S.Wu,F.Crestani,F.Gibb}@cis.strath.ac.uk

Abstract. In distributed information retrieval systems, document overlaps occur frequently across results from different resources. This is especially the case for meta-search engines which merge results from several web search engines. This paper addresses the problem of merging results exploiting overlaps in order to achieve better performance. New algorithms for merging results are proposed, which take advantage of the use of duplicate documents in two ways: one correlates scores from different results; the other regards duplicates as increasing evidence of being relevant to the given query. An extensive experimentation has demonstrated that these methods are effective.

1 Introduction

With the widespread use of wide area networks and especially the Internet, online information systems proliferate very rapidly. Users often find it necessary to search different resources to satisfy an information need. In addition, for information that is proprietary, costs money, or its publisher wishes to control it carefully, it cannot be collected and indexed in a centralised database environment. In such cases, distributed information retrieval systems become an alternative solution, and one in which merging of results is an aspect which demands careful attention.

In a distributed information retrieval (DIR) system we have resource descriptions that contain meta-data describing the main characteristics of the resources. Typically, a DIR system involves the following major steps:

1. A broker, responsible for managing the query process, receives a user query from a client and selects a subset of resources which can best satisfy the user's information need according to the resource descriptions used by the broker (resource selection).
2. The broker sends the query to all the selected resources and collects the results from them.
3. The broker merges these results into a single list (results merging or data fusion). How to rank these documents in the list is an important issue.
4. The broker sends the merged result back to the client, which displays them to the user.

For merging of results, three cases can be identified depending on the degree of overlap among the selected databases for a given query [12]:

1. the databases are pairwise disjoint or nearly disjoint;
2. the databases overlap but are not identical;
3. the databases are identical.

Case 1 and case 2 may occur quite often in the DIR environment. For example, a group of special-purpose web search engines may have very little overlaps among them, while several general-purpose web search engines may have considerable overlaps. Case 3 usually does not occur in the typical DIR environment. However, it could be exploited to improve retrieve effectiveness.

Both cases 1 and 3 have been discussed extensively by many researchers, for example, in [2,4,5,6,10,18,20,21,23]. However, results merging algorithms suitable for Case 2 remains an open question [12]. Previously, meta search engines treated this in two typical ways. One regards duplicates as redundant information, which are discarded when we find out that they exist. Such a method was used by Profusion [7,16]. The other is using methods such as CombSum, which are suitable for Case 3. The latter method was used by Metacrawler in its early stage [13,17,18] [1]. Neither of the above solutions is a good solution. The first does not use the information implied by the duplicates. The second ignores the difference between identical databases and partially overlapping databases, so that methods such as linear combination are not appropriate in this case. We will return on the differences between these methods later on in the paper.

We argue that results merging in case 2 is different and more complicated than that in case 1 or case 3. In this paper, we present a group of algorithms which are specifically designed for case 2. Our solution integrates the techniques used for case 1 and case 3.

The rest of the paper is organised as follows. In Section 2 we review some related work on data fusion with totally identical databases and results merging without overlapping. Section 3 presents new methods for merging results from overlapping databases. The experimental setting and results are presented in Sections 4 and 5. Section 6 concludes the paper.

2 Previous Work on Results Merging

There has been a lot of research on results merging with identical databases, which is called data fusion in many publications. Fox and Show [6] proposed a number of combination techniques including operators like Min, Max, CombSum and CombMNZ. CombSum sets the score of each document in the combination to the sum of the scores obtained by the individual resource, while in CombMNZ

[1] In addition,there are methods that download all the documents in the result sets and re-rank them by running a local retrieval system or an analysis tool. In this paper we do not consider such methods.

the score of each document was obtained by multiplying this sum by the number of resources which had non-zero scores. Note that summing (CombSum) is equivalent to averaging while CombMNZ is equivalent to weighted averaging.

Lee [10] studied this further with six different servers. His contribution was to normalise each information retrieval server on a per-query basis which improved results substantially. Lee showed that CombMNZ worked best, followed by CombSum while operators like Min and Max were the worst.

Vogt and Cottrell [21] proposed an approach based on a linear combination of scores. In this approach, the relevance of a document to a query is computed by combining both a score that captures the quality of each resource and a score that captures the quality of the document with respect to the query. Formally, if q is a query, d is a document, s is the number of resources, and $w = (w_1, w_2, ..., w_s)$ are the resource scores, then the overall relevance p of d in the context of the combined list is given by $p(w, d, q) = \sum_{i=1}^{s} w_i p_i(d, q)$.

For non-overlapping results from different resources, Round-Robin is a simple but effective merging method, which takes one document in turn from each available result set. However, the effectiveness of such a method depends on the performances of the component resource servers. If all the results have similar effectiveness, then Round-Robin performs well; if some of the results are very poor, then Round-Robin becomes very poor too.

Voorhees, Gupta and Johnson-Laird [20] demonstrated a way of improving the Round-Robin method. By running some training queries, they estimated the performance of each resource. When a new query is encountered, they retrieve a certain number of documents from each resource based on its estimated performance.

Callan et al. [2] proposed a merging strategy based on the scores achieved by both resource and document. The score for each resource is calculated by a resource selection algorithm, CORI, which evaluates the “goodness” of a resource with respect to a given query among a set of available resources. The score of a document is the value that document obtains from that resource, which indicates the “goodness” of that document to a given query among a set of retrieved documents from that resource.

Calve and Savoy [3] proposed a merging strategy based on logistic regression. Some training queries are needed for setting up the model. Their experiments show that the logistic regression approach is significantly better than Round-Robin, raw-score, and normalised raw-score approaches.

Si and Callan [19] used a linear regression model for results merging, with an assumption that there is a centralised sample database which stores a certain number of documents from each resource. The DIR broker runs the query on the sample database at the same time it is sent to the different resource servers. It then finds the duplicate documents in the results for the sample database and for each resource, and normalises the scores of these documents in all results by a linear regression analysis with the document scores from the sample database serving as a baseline. Their experimental results show that their method is slightly better than CORI.

Finally, let us review the results merging policies used by some meta search engines. Metacrawler [13,17,18] is probably one of the first meta search engines that were developed for the Web. Its results merging method was relatively simple, eliminating duplicate URLs first then merging the results by using the CombSum function. Profusion [7,16], another early meta search engine, merged all duplicate URLs by using the Max function. Savvy Search [5,8] focused primarily on the problem of learning and identifying the right set of search engines to which to issue the queries, and to a lesser extent on how the returned results are merged. Inquirus [9] took a different approach to the problem. Instead of relying on the engine's ranking mechanism, it retrieved the full contents of the web pages returned and ranked them more like a traditional search engine, using information retrieval techniques applicable to full documents. Similar to Inquirus, Mearf [11,15] retrieved the full contents of the web pages returned, then re-ranked them based on the analytical results on contents and/or links of these web pages.

Though regression analysis has been used previously both Calve and Savoy [3] and Si and Callan [19], we use this method in a different situation, where partial overlaps exist between different resources. In our case, no training is needed as in Calve and Savoy's work, and neither a centralised sample database nor an information retrieval system are needed in the DIR broker as in Si and Callan's work. How to merge documents which appear in only one result with those which appear in several different results is a new issue, which has not been considered before.

3 Results Merging with Partially Overlapping Databases

In this paper, we assume that we have a group of resources which are partially overlapping. For a given query, we obtain results from these resources. Except that, no description information about these resources is required. In addition, we do not consider downloading all the documents from different resources for re-ranking.

Usually, we do not need to check the full texts of all the documents in order to identify many of the duplicates. For example, in most cases the URL can be used as an identifier for every document in web search engines. Despite its limitations such technique is still currently used by many search engines. Alternatively, we may estimate that two documents are the same by comparing their meta-data, such as the titles, authors, publishers, and so on. Besides, one assumption we make is that all documents in the results have scores which have been obtained from their resource servers.

From overlapping databases, we have more information (the document duplicates) to merge results than from pairwise disjoint databases. We may use this information in two different ways. First, the duplicate document pairs in two different results may be helpful for correlating the scores of all the documents contained in two different results. As in [3,19], regression analysis is a useful tool for achieving this. On the other hand, a document appearing in more than one result can be regarded as increasing evidence of being relevant to the given query. However, how to merge documents which occur only in one result with those which occur in more than one result is a new issue. Some of the techniques [6,10,21] employed in data fusion can be used here. However, there are some important differences which need to be considered.

In data fusion one assumption is: the more often a document occurs in different results, the more likely it is a relevant document to the given query. However, when different resource servers retrieve different databases with some overlaps with each other, the above assumption can not be applied directly. For example, suppose that for a given query q, resource server A retrieves documents d_1 and d_3, resource server B retrieves documents d_2 and d_3. Thus, d_3 occurs in both results of A and B, but d_1 and d_2 only occur once. Does that suggest d_3 is more likely to be relevant to the query than d_1 and d_2? We are unable to reach such a conclusion unless we know specifically that d_1 is in B's database and d_2 is in A's database. However, we may not have this information and in Section 3.2 we discuss this in more detail.

3.1 Regression Analysis

Regression analysis is at the basis of our results merging methods. For scores of the same documents in two different results, we assume that a linear relationship exists between them (x_i and y_i):

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

We can use the least squares point estimates b_1 and b_0 for parameters β_1 and β_0. In the simple linear regression model they are calculated using the following formulae [1]:

$$b_1 = \frac{n\sum_{i=1}^{n} x_i y_i - (\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} y_i)}{n\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2}$$

and

$$b_0 = \frac{\sum_{i=1}^{n} y_i}{n} - \frac{b_1(\sum_{i=1}^{n})x_i}{n}$$

If only two systems are involved, then we need only do it once. If n ($n > 2$) resource servers are involved, we assume that for each of them, there are overlaps with at least one other resource. If this cannot be satisfied for any resource, we can use the method proposed by Si and Callan [19], which was mentioned in Section 2 of this paper.

We will have to do $n - 1$ regression analyses, each with a pair of resources. The following algorithm can be used for selecting the pairs:

1. assign every retrieval system to an isolated node;
2. check all pairs of nodes which have no connection (direct or indirect) and have the most duplicate documents; make a connection between these two nodes;
3. repeat Step 2 until all nodes are connected.

For every selected pair of systems, we carry out a regression analysis. After $n-1$ pairs, we can get a normalised score for every document in every system without any inconsistency at any circumstance. We assume that the more duplicate documents are involved, the more accurate the regression analysis is, that is why we select the pair with the most duplicate documents in the above algorithm. If the result returned by one server has no duplicates with any other servers, then we can use the method proposed by Si and Callan [19] to deal with it.

3.2 Results Merging by Shadow Document

After performing regression analysis for each pair of results, we normalise those scores from different resources and make them comparable. Now the remaining problem is how to deal with the duplicate documents for better performance. One solution is to take the maximum score among the duplicate documents as the score of that document in the fused result, another solution is to take the minimal score, the third is to average them. Then we can merge these documents by descending order on scores. These methods will be referred to as Max, Min, and Avg. Obviously these methods do not use overlaps as increasing evidence of relevance to the query.

Next let us introduce some more advanced methods. Suppose we have two resource servers A and B. If document d only occurs in A's result but not in B's result, there are two possibilities: (1) d is not stored in B's database; (2) d is not retrieved by B. In case 1, we guess that if d was in B's database, then B would assign d a score similar to $s_a(d)$. In case 2, d must have obtained a very low score since it does not occur in the result. The possibility of case 2 is due to the extent of overlaps existing in A and B. However, for any particular document, we cannot judge which situation is happening.

For a given query, if document d occurs in both A and B's results then we sum the two scores as d's total score; if d only occurs in A's result with a score $s_a(d)$, we pretend that there is a shadow document of it in B, and assign a score to (shadow) document d in B. We do the same for those documents which only occur in B's result but not in A's result. Then we can use the same method as above (sum) to merge all the results. More specifically, if document d only occurs in A's result, d's total score is calculated as $s_a(d)(2-k_1)$, where k_1 is a coefficient ($0 \leq k_1 \leq 1$). The situation is similar for documents only occurring in B's result. The assignment of an optimal values to k_1 is important. In the our experimentation the value will be determined empirically , although an analysis of its influence on the performance of the models will help knowing its influence.

Some modifications can be done to the above method. One way is to let the coefficient k_1 be related to the overlapping rate of two databases. We need some statistical information about these databases (e.g., size, overlapping rate), and methods such as the one reported in [22] can be used. However, to collect such information usually takes a lot of effort. Here we assume that this information is not available. For a given query, we obtain different results from different resources. We calculate the number of duplicate documents between each pair of them, then use this information as the indicator of the overlapping rate of these two databases. Suppose n is the number of documents included in both results and m is the number of duplicate documents. A higher overlapping rate suggests that there are more duplicate documents in both databases. We can rely on this to modify the above method. If document d only occurs in A, then d's total score is calculated as $s_a(d)(2 - k_2 * (1 - m/n))$, where k_2 is a coefficient ($0 \le k_2 \le 1$). As for k_1, we need to carry out some experiments to empirically find the best value for k_2. In the following these two methods will be referred to as Shadow Document Method (SDM) 1 and 2. Obviously, we can easily extend these SDMs with more than 2 resource servers.

4 Experimental Settings

Two groups of full-text resources were used for the experiments. Each group included 3 resources. All of them were subsets of the first two CDROMs of the TREC Collection. The characteristics of these resources are presented in Table 1. In group 1, WSJ(87-89) and WSJ(88-91) have 52284 documents in common, WSJ(88-91) and WSJ(91-92) have 42465 documents in common. Clearly, in both cases, considerable overlaps exist. In group 2, each database includes the same 6103 documents from WSJ90. In addition AP(88-89)+ includes 115143 documents from AP(88-89), WSJ(87-89)+ includes 98732 documents from WSJ(87-89), and FR(88-89)+ includes 45820 documents from FR(88-89)[2]. The percentage of overlap in AP(88-89)+, WSJ(87-89)+, and FR(88-89)+ are 5%, 6%, and 12%, respectively. The two groups are substantially different. Group 1 is more homogenous and presents a heavy overlap among databases. Group 2 is more heterogenous and presents a light overlap among databases. Arguably, Group 2 could be considered as a more realistic testbed for results merging than Group 1.

Table 1. Resources used in the experimentation.

Group	Resources	Size	Num. of docs	Retrieval model
	WSJ(88-91)	358 MB	116641	vector space model
1	WSJ(87-89)	273 MB	98732	okapi model
	WSJ(91-92)	176 MB	52815	language model
	AP(88-89)+	371 MB	121246	vector space model
2	WSJ(87-89)+	293 MB	104835	okapi model
	FR(88-89)+	500 MB	51923	language model

In both groups, the result merging problem has been simulated by producing ranking using a different retrieval system for each resource. The Lemur information retrieval toolkit [14], which includes three different retrieval models (the

[2] The symbol "+" indicates that the named collection includes the 6103 documents of the WSJ90.

vector space model, the probabilistic model okapi, and the language model), was used to implement the retrieval system. Table 1 indicates which model has been in relation to each resource.

Average precision figures were evaluated using the standard definition of precision at different document ranks levels. In all cases, 100 queries (TREC topics 101-200) with their respective relevance assessments were used for the experiments.

Two methods were used as baselines. One was the Round-Robin approach. The other was "the whole database" approach. Here "the whole database" approach means that we use only one information retrieval system to retrieve from a database that includes all the documents contained in the different resources.

We tested the effectiveness of our SDMs for both heavily and lightly overlapping situations. In addition, we compared the performances of CombMNZ, CombSum, and SDM methods by using Group 1 databases, that is in conditions of heavy overlapping.

5 Experimental Results

Here we present some of the results of an extensive experimentation on the performance of the proposed results merging methods. Only the most relevant results are presented. We present the experimental results with heavy and light overlaps in Sections 5.1 and 5.2, respectively. In Section 5.3 we compare the performances of SDM and CombSum and CombMNZ. A detailed discussion follows in Section 5.4 which explains we found that CombSum and CombMNZ perform considerably worse than SDM in our experiment. In Section 5.5 Enhanced SDMs are introduced and the corresponding experimental results are presented.

5.1 Results Merging with Heavy Overlaps

Three databases, which are indicated in Table 1 as group 1, were used for the experiment. For each query, each information retrieval system retrieves 1000 documents with scores. Different merging methods were experimented with for these result sets. The experimental results are shown in Table 2, in which different methods are compared with Round-Robin, and the difference between them is shown in parentheses. For the two SDMs, k_1 is set to be 0.05 and k_2 is set to be 0.1, which are the optimum empirical values observed in our experiments. The effect of the coefficient values on the SDMs will be discussed later. The experimental results show that all methods are better than Round-Robin. In particular, Ave, Max, and Min are very similar in performance, and the two SDMs are very similar in performance as well.

Because we use the duplicate documents for the regression analysis to normalise the scores in different results, this makes Ave, Max, and Min much similar. They outperform Round-Robin by about 7%. SDMs outperform Round-Robin by about 9%, and outperform Ave, Max, and Min by about 2%.

Table 2. Precision comparison of different methods with heavy overlaps in databases (WSJ(88-91), WSJ(87-89), WSJ(91-92)).

Docs Rank	Round -Robin	Avg	Max	Min	SDM1 (k=0.05)	SDM2 (k=0.1)
5	0.4633	0.5143 (+11.0%)	0.5102 (+10.1%)	0.5143 (+11.0%)	0.5283 (+14.0%)	0.5306 (+14.5%)
10	0.4378	0.4704 (+7.4%)	0.4714 (+7.7%)	0.4745 (+8.4%)	0.4817 (+10.0%)	0.4806 (+9.8%)
15	0.4014	0.4401 (+9.6%)	0.4375 (+9.0%)	0.4402 (+9.7%)	0.4409 (+9.8%)	0.4402 (+9.7%)
20	0.3822	0.4102 (+7.3%)	0.4097 (+7.2%)	0.4092 (+7.1%)	0.4214 (+10.3%)	0.4194 (+9.7%)
25	0.3649	0.3943 (+8.1%)	0.3902 (+6.9%)	0.3939 (+7.9%)	0.3976 (+9.0%)	0.3980 (+9.1%)
30	0.3514	0.3783 (+7.7%)	0.3725 (+6.3%)	0.3769 (+7.3%)	0.3793 (+7.9%)	0.3800 (+8.1%)
50	0.3106	0.3292 (+6.0%)	0.3296 (+6.1%)	0.3288 (+5.9%)	0.3333 (+7.3%)	0.3343 (+7.6%)
100	0.2426	0.2533 (+4.4%)	0.2530 (+4.3%)	0.2518 (+3.8%)	0.2571 (+6.0%)	0.2572 (+6.0%)

Also, we compared SDM2 with the single database setting. We set up a single database that includes all the documents contained in the 3 resources (WSJ87-92), then used the three models (vector space, okapi, and language model) in Lemur to retrieve documents from it. The experimental results, reported in Table 3, show that SDM2 is a little better than Okapi, and much better than the Language model, but a little worse than the vector space model. Similar results were obtained for SDM1.

Table 3. Comparison of SDM with single database setting when heavy overlaps exist among databases (WSJ(88-91), WSJ(87-89), WSJ(91-92)).

Docs Rank	SDM2	Okapi model	Vector space model	Language model
5	0.5306	0.4960 (-6.5%)	0.5326 (+0.3%)	0.4600 (-13.3%)
10	0.4806	0.4650 (-3.2%)	0.4755 (-1.1%)	0.4420 (-8.0%)
15	0.4402	0.4394 (-0.2%)	0.4415 (+0.3%)	0.4213 (-4.3%)
20	0.4194	0.4170 (-0.6%)	0.4379 (+4.4%)	0.4070 (-3.0%)
25	0.3980	0.3968 (-0.3%)	0.4276 (+7.4%)	0.3888 (-2.3%)
30	0.3800	0.3784 (-0.4%)	0.3915 (+3.0%)	0.3790 (-0.3%)
50	0.3343	0.3246 (-2.9%)	0.3365 (+0.7%)	0.3300 (-1.3%)
100	0.2565	0.2532 (-1.3%)	0.2565(±0.0%)	0.2572 (+0.3%)

Different coefficients were tested for the two SDMs. The experimental results are shown in Figure 1. For SDM1, the maximum value occurs when k_1 is between

0.05 and 0.1, then the average precision drops quite quickly. For SDM2, the curve is rather flat from the beginning to the end, which indicates that SDM2 is more robust to different values of k. Since, in many cases, the overlapping rate for a query result is around 300/1000, a value of 0.8 for k_2 is roughly equivalent to a value of 0.24 for k_1. However, it seems that in both cases, a very small coefficient may improve the performance as effectively as a larger one.

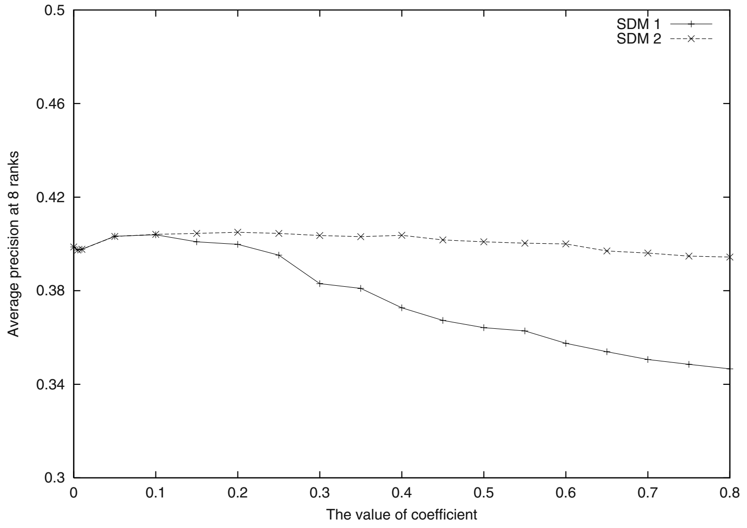


Fig. 1. Performance of two SDMs with different coefficients when heavy overlaps exist among databases (WSJ(88-91), WSJ(87-89), and WSJ(91-92).

5.2 Results Merging with Light Overlaps

In this experiment, only light overlaps exist among the different databases. The three databases indicated as group 2 in Table 1 were used. The represent a more common measure of overlap since, very often, there are only about 10-20 duplicate documents in different results. In such a case, all 5 merging methods behave very similarly, as expected. The reason is that the number of duplicate documents is very small, so it may not be possible for the two SDMs to outperform Min, Max, or Avg. The experimental results are shown in Tables 4 and 5. The results reported in Table 4 show that all merging methods are considerably better than Round-Robin (over 10% improvement), and they are as good as the single database setting, as reported in 5.

Figure 2 shows the experimental results for two SDMs when different coefficients are assigned. The SDM1 curve decreases when the value of coefficient increases, while SDM2 is almost totally flat all the way from the beginning to the end, showing that SMD2 is almost completely independent from the value of k.

In the above we have presented experimental results of two SDMs in two typical situations: either heavy overlap or light overlap exists in different results. SDM1 gets its maximum performance when k_1 is between 0.05 and 0.1 for heavy overlaps, while it gets its maximum performance when k_1 takes a value of 0.01 or even less for light overlaps. For SDM2, a value of 0.1 seems to fit well for both situations. It suggests that for SDM1, it is better to select a different value of k_1 for different overlap rates, while for SDM2, a fixed value of 0.1 for k_2 is very likely to be fine for all situations.

Table 4. Precision comparison of different methods with light overlaps in databases (AP(88-89)+, WSJ(87-89)+, and FR(88-89)+).

Docs Rank	Round -Robin	Avg	Max	Min	SDM1 (k=0.05)	SDM2 (k=0.1)
5	0.4504	0.5024 (+11.5%)	0.4964 (+10.2%)	0.4984 (+10.66%)	0.5004 (+11.1%)	0.4942 (+9.7%)
10	0.4069	0.4652 (+14.3%)	0.4652 (+14.3%)	0.4663 (+14.6%)	0.4642 (+14.1%)	0.4672 (+14.8%)
15	0.3762	0.4434 (+17.9%)	0.4421 (+17.5%)	0.4421 (+17.5%)	0.4421 (+17.5%)	0.4420 (+17.5%)
20	0.3639	0.4179 (+14.8%)	0.4173 (+14.7%)	0.4184 (+15.0%)	0.4173 (+14.7%)	0.4224 (+16.1%)
25	0.3389	0.3956 (+16.7%)	0.3944 (+16.4%)	0.3944 (+16.4%)	0.3936 (+16.1%)	0.3944 (+16.4%)
30	0.3368	0.3917 (+16.3%)	0.3902 (+15.9%)	0.3819 (+13.4%)	0.3819 (+13.4%)	0.3825 (+13.6%)
50	0.2962	0.3443 (+16.2%)	0.3436 (+16.0%)	0.3441 (+16.2%)	0.3442 (+16.2%)	0.3377 (+14.1%)
100	0.2428	0.2818 (+16.1%)	0.2811 (+15.8%)	0.2812 (+15.8%)	0.2817 (+16.0%)	0.2788 (+14.8%)

5.3 Comparison of SDMs with CombMNZ and CombSum

When different databases have considerable overlaps, CombMNZ and CombSum have show to be effective results merging methods. We therefore decided to compare SDMs with CombMNZ and CombSum using the heavy overlapping databases of Group 1. CombMNZ and CombSum can be used with two different normalisation methods: linear regression and linear [0,1] normalisation [10]. We have described linear regression in Section 3. Linear [0,1] is the usual normalisation method for CombMNZ and CombSum. For a list of documents from a resource, we use the following formula to calculate the score for any document in the list: $norm\_score = \frac{score-min\_score}{max\_score-min\_score}$, where max_score denotes the highest score in the document list, min_score denotes the lowest score in the

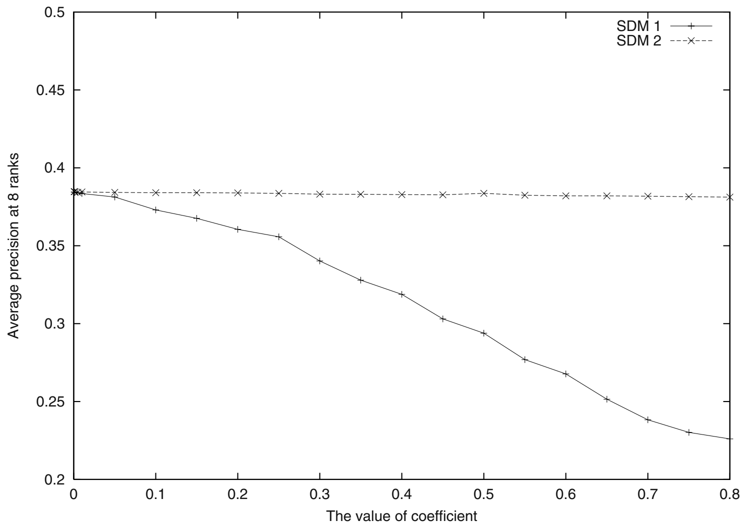


Fig. 2. Performance of two SDMs with different coefficients when light overlaps exist among databases (AP(88-89)+, WSJ(87-89)+, and FR(88-89)+)

Table 5. Comparison of SDM with single database setting when light overlaps exist among databases (AP(88-89)+, WSJ(87-89)+, FR(88-89)+).

Docs Rank	SDM2	Okapi model	Vector space model	Language model
5	0.4942	0.5040 (+2.0%)	0.5024 (+1.7%)	0.4721 (-4.5%)
10	0.4672	0.4650 (-0.4%)	0.4532 (-3.0%)	0.4359 (-6.7%)
15	0.4420	0.4420 (±0.0%)	0.4293 (-2.9%)	0.4326 (-2.1%)
20	0.4224	0.4210 (-0.3%)	0.4058 (-3.9%)	0.4138 (-2.0%)
25	0.3944	0.4072 (+3.3%)	0.3902 (-1.1%)	0.3988 (+1.1%)
30	0.3825	0.3990 (+4.3%)	0.3723 (-2.7%)	0.3902 (+2.0%)
50	0.3377	0.3526 (+4.7%)	0.3411 (-1.0%)	0.3543 (+4.9%)
100	0.2788	0.2830 (+1.5%)	0.2742 (-1.6%)	0.2835 (+1.7%)

document list, *score* denotes the score of a given document, and *norm_score* is the normalised score of the given document. Thus, the document with the highest score is assigned a score of 1, the document with the lowest score is assigned a score of 0, any other document is assigned a score between 0 and 1.

Experimental results of CombMNZ and CombSum are shown in Table 6, where they are compared with SDMs. For all of them, SDM1 with $k_1 = 0.05$ serves as the baseline. The results show that both CombMNZ and CombSum are considerably worse than SDM1. Using either linear [0,1] normalisation or linear regression normalisation, both CombMNZ and CombSum are about 10% to 20% worse than SDM1.

Table 6. Precisions of CombMNZ and CombSum with heavy overlaps in databases (WSJ(88-91), WSJ(87-89), and WSJ(91-92)). SDM1 serves as the baseline.

Docs Rank	SDM1 (k=0.05)	CombMNZ (Regression)	CombSum (Regression)	CombMNZ (Linear[0,1])	CombSum (Linear[0,1])
5	0.5283	0.4740	0.4740	0.4660	0.4660
		(-10.3%)	(-10.3%)	(-11.8%)	(-11.8%)
10	0.4817	0.4150	0.4150	0.4120	0.4160
		(-13.8%)	(-13.8%)	(-14.5%)	(-13.6%)
15	0.4409	0.3660	0.3707	0.3780	0.3874
		(-17.0%)	(-15.9%)	(-14.3%)	(-12.1%)
20	0.4214	0.3460	0. 3530	0.3455	0.3580
		(-17.9%)	(-16.2%)	(-18.0%)	(-15.0%)
25	0.3976	0.3264	0.3348	0.3292	0.3444
		(-17.9%)	(-15.8%)	(-17.2%)	(-13.4%)
30	0.3793	0.3073	0.3177	0.3117	0.3297
		(-19.0%)	(-16.3%)	(-17.8%)	(-13.1%)
50	0.3333	0.2562	0.2668	0.2652	0.2846
		(-23.1%)	(-20.0%)	(-20.4%)	(-14.6%)
100	0.2571	0.1886	0.2010	0.2006	0.2284
		(-26.6%)	(-21.8%)	(-22.0%)	(-11.2%)

5.4 Analysis

If only very few duplicates occurs in different results, linear regression is a very good method for normalising scores, using either Max, Min, Avg or SDMs can achieve a very good merging results. However, when there are considerable overlaps it is a different matter. In this case we have shown that SDMs perform better that CombMNZ and CombSum. Let us try to explain why.

With normalised score, it is possible for us to compare the scores of documents in different results. However, we have to distinguish several different situations. Let us consider the situation with two databases A and B again. (1) a document d only occurs in the result of A or B; (2) a document d occurs in the results of both A and B. We can divide documents into two groups “1db” and “2db” according to this. In the following, when we talk about the score of a document, it always means the normalised score.

For documents in 1db, whatever method we use (CombMNZ, CombSum, or SDMs), we just get the same order in the merging, as long as we use the same normalisation method for them. It is the same in 2db. If we have two documents d_1 and d_2, and the sum of d_1's two scores is lower than the sum of d_2's two scores, then d_1 is always ranking lower than d_2 in all merging methods. The only difference among these methods lies in ranking documents in 1db with those in 2db. For CombSum and CombMNZ, we always rank documents in 1db very low. This is correct for 1db documents, since we know that the document will not be retrieved by any other resources. However, if these databases are partially overlapping, we do not know if the document is not retrieved or is not stored in that database at all, so CombSum and CombMNZ become questionable because they over-promote documents that occur in both A and B. On the other hand, SDMs behave in both situations.

In group 1, WSJ88, 89, and 91 were stored in two databases, which totals 94,749 documents; while WSJ87, 90, and 92 were stored in one database, which totals 78,690 documents. For all queries (TREC101-200), usually about 500 documents were retrieved by two databases, the rest (about 2500) were retrieved by only one database. Table 7 lists the precision of each of these two groups.

Table 7. Precision of retrieving at different document levels, grouped by the number of databases retrieving the documents. Three databases used are WSJ(88-91), WSJ(87-89), and WSJ(91-92).

Retrieved	Precision at different document levels							
group	5	10	15	20	25	30	50	100
1db	0.4468	0.4001	0.3665	0.3431	0.3169	0.2988	0.2439	0.1671
2db	0.4740	0.4150	0.3660	0.3460	0.3264	0.3073	0.2558	0.1874

Using CombMNZ, all of the top 30 documents and over 95% of the top 100 documents in the fused result are of 2db group. That is why its performance is almost totally determined by the precision of the 2db group. CombSum is a little better than CombMNZ, but the 2db group is still in the dominant position. SDMs roughly took about 60% of the documents from 2db and about 40% of the documents from 1db. On the one hand, such a balanced policy improves the precision; on the other hand, fused result with this method has a more wide coverage. If using CombMNZ or CombSum, the 1db group, which is composed of almost half of the total documents, is almost completely ignored.

5.5 Enhanced Shadow Document Methods

If we had more information about databases and documents, could we make SDM1 and SDM2 perform better? Suppose for any given document, we could know if it is stored in a given database. Such a situation is not unusual in corporate networks, for example. Similarly, we could identify the source of a document by checking its identifier, so we could know which document is present in more than one site.

Let us discuss this with two databases A and B. For a given query Q, if a document d only occurs in the result from A but not B, one of two possible things must have happened. (1) d is not stored in B; (2) d is stored in B but not retrieved. If the distributed information retrieval system is able to distinguish which situation has happened, then we could modify the two SDM methods in the following way: if d is not stored in B, then we just treat it as before; if d is stored in B but not retrieved, then the document d only gets itths score (normalised) from A, but not any extra score from B. We refer to this modification of SDM as Enhanced SDM (ESDM).

The experiment was carried out with databases in group 1. The experimental result is shown in Table 8, where ESDM1 is compared with SDM1, and ESDM2 is compared with SDM2. The experimental result shows that on the top 5 to 10 documents, both ESDM1 and ESDM2 are slightly (but significantly) better than their counterparts, while in all other rank levels they perform in a very similar way.

Table 8. Precision comparison of SDMs with different information and three databases (WSJ(88-91), WSJ(87-89), and WSJ(91-92)). The figures in parentheses indicate the difference between that method and its counterpart with normal information.

Docs Rank	SDM1	ESDM1	SDM2	ESDM2
5	0.5004	0.5246 (+4.8%)	0.4942	0.5154 (+4.1%)
10	0.4642	0.4765 (+2.6%)	0.4672	0.4733 (+2.4%)
15	0.4421	0.4470 (+1.1%)	0.4420	0.4461 (+0.9%)
20	0.4173	0.4214 (+1.0%)	0.4224	0.4218 (-0.1%)
25	0.3936	0.3997 (+1.5%)	0.3944	0.3969 (+0.6%)
30	0.3819	0.3825 (+0.2%)	0.3825	0.3812 (-0.3%)
50	0.3442	0.3394 (-1.4%)	0.3377	0.3283 (-1.4%)
100	0.2817	0.2735 (-2.9%)	0.2788	0.2753 (-1.2%)

6 Conclusions

In this paper we have presented several methods for results merging when overlaps exist between different results in a distributed information retrieval system. These methods use linear regression analysis and shadow document merging method to improve the performance. Experiments have been conducted in two typical situations: heavy overlapping and light overlapping. In the former situation, two shadow document methods perform slightly better than Min, Max, and Avg, and are slightly better than the single database setting for some retrieval models. Experiments also show that in all situations, all methods proposed are more effective than Round-Robin by over 10% on average.

With heavy overlapping, we compared the performances of CombMNZ and CombSum and SDM. The experimental result shows that both CombMNZ and CombSum are 10% to 20% worse than SDM.

When we have more information about the documents, that is if we can identify for any document if it is stored in any given database, then SDMs can work a little better for the top 5 to 10 documents when there are heavy overlaps among databases.

Since overlap happens frequently in general web search engines, the proposed methods in this paper are very desirable for merging web search results. Compared with those methods presented in [9,17,18] our methods can achieve good results but without the effort of downloading web pages and local retrieval process.

Considering in distributed information retrieval systems, a relatively large number of systems could be involved. Further experiments are required to evaluate the performances of proposed methods at such circusmstance.

Acknowledgements. This work is supported by the EU Commission under IST Project MIND (IST-2000-26061). More information about MIND can be found at `http://www.mind-project.net/`.

References

1. B. L. Bowerman and R. T. O'Connell. *Linear Statistical Models: An Applied Approach*. PWS-KENT Publishing Company, 1990.
2. J. K. Callan, Z. Lu, and W. Croft. Searching distributed collections with inference networks. In *Proceedings of the 18th Annual International ACM SIGIR Conference*, pages 21–28, Seattle, USA, July 1995.
3. A. L. Calve and J. Savoy. Database merging strategy based on logistic regression. *Information Processing and Management*, 36(3):341–359, 2000.
4. N. Craswell, P. Bailer, and D. Hawking. Server selection on the world wide web. In *Proceedings of the Fifth ACM Conference on Digital Libraries*, pages 37–46, San Antonio, CA, USA, June 2000.
5. D. Dreilinger and A. Howe. Experiences with selectiong search engines using metasearch. *ACM Transaction on Information Systems*, 15(3):195–222, 1997.
6. E. A. Fox and J. Shaw. Combination of multiple searchs. In *The Second Text REtrieval Conference (TREC-2)*, pages 243–252, Gaitherburg, MD, USA, August 1994.
7. S. Gauch, G. Wang, and M. Gomez. Profusion: Intelligent fusion from multiple, distributed search engines. *Journal of Universal Computer Science*, 2(9):637–649, September 1996.
8. A. Howe and D. Dreilinger. Savvysearch: A meta-search engine that learns which search engines to query. *AI Magazine*, 18(2):19–25, 1997.
9. Inquirus. http://www.inquirus.com/.
10. J. H. Lee. Analysis of multiple evidence combination. In *Proceedings of the 20th Annual International ACM SIGIR Conference*, pages 267–275, Philadelphia, Pennsylvania, USA, July 1997.
11. Mearf. http://mearf.cs.umn.edu/.
12. W. Meng, C. Yu, and K. Liu. Building efficient and effective metasearch engines. *ACM Computing Surveys*, 34(1):48–89, March 2002.
13. Metacrawler. http://www.metacrawler.com/.
14. P. Ogilvie and J. Callan. Experiments using the lemur toolkit. In *Proceedings of the 2001 Text REtrieval Conference*, pages 103–108, Gaithersburg, Maryland, USA, November 2001.
15. B. Oztekin, G. Karypis, and V. Kumar. Expert agreement and content baed reranking in a meta search environment using mearf. In *Proceedings of the 11th International World Wide Web Conference*, pages 333–344, Honolulu, Hawaii, USA, May 2002.
16. Profusion. http://www.profusion.com/.
17. E. Selberg and O. Etzioni. Multi-service search and comparison using the metacrawler. In *Proceedings of the 4th International World Wide Web Conference*, Paris, France, May 1996.

18. E. Selberg and O. Etzioni. The metacrawler architecture for resource aggregion on the web. *IEEE Expert*, 12(1):8–14, 1997.
19. L. Si and J. Callan. Using sampled data and regression to merge search enging results. In *Proceedings of the 25th Annual International ACM SIGIR Conference*, pages 19–26, Tempere, Finland, August 2002.
20. E. M. Voorhees, N. K. Gupta, and B. Johnson-Laird. Learning collection fusion strategies. In *Proceedings of the 18th Annual International ACM SIGIR Conference*, pages 172–179, Seattle, Washington, USA, July 1995.
21. C. C. Vort and G. A. Cottrell. A fusion via a linear combination of scores. *Information Retrieval*, 1(3):151–173, 1999.
22. S. Wu, F. Gibb, and F. Crestani. Experiments with document archive size detection. In *Proceedings of the 25th European Conference on Information Retrieval Research*, pages 294–304, Pisa, Italy, April 2003.
23. C. Yu, G. Meng, K. Liu, W. Wu, and N. Rishe. Efficient and effective metasearch for a large number of text databases. In *Proceedings of the Eighth ACM International Conference on Information and Knowledge Management*, pages 217–224, Kansas City, USA, November 2001.

Recent Results on Fusion of Effective Retrieval Strategies in the Same Information Retrieval System*

Steven M. Beitzel, Eric C. Jensen, Abdur Chowdhury, David Grossman, Nazli Goharian, and Ophir Frieder

Information Retrieval Laboratory
Department of Computer Science
Illinois Institute of Technology
Chicago, IL 60616
{steve,ej,abdur,dagr,nazli,ophir}@ir.iit.edu

Abstract. Prior efforts have shown that data fusion techniques can be used to improve retrieval effectiveness under certain situations. Although the precise conditions necessary for fusion to improve retrieval have not been identified, it is widely believed that as long as component result sets used in fusion have higher relevant overlap than non-relevant overlap, improvements due to fusion can be observed. We show that this is not the case when systemic differences are held constant and different highly effective document retrieval strategies are fused within the same information retrieval system. Furthermore, our experiments have shown that the ratio of relevant to non-relevant overlap is a poor indicator of the likelihood of fusion's effectiveness, and we propose an alternate hypothesis of what needs to happen in order for fusion to improve retrieval when standard voting/merging algorithms such as CombMNZ are employed.

1 Introduction

Recently there has been much research done in the field of information retrieval concerning the various kinds of "fusion" and their applications. Data fusion is the combination of multiple pieces of evidence of relevance, such as different query representations, different document representations, and different retrieval strategies used to obtain a measure of similarity between a query and a document. This combination is then typically utilized to improve retrieval effectiveness, and is most often applied to the task of ad-hoc retrieval. Data fusion techniques also have applications outside the realm of ad hoc retrieval, having a relevant place in the worlds of metasearch, and distributed information retrieval.

This paper summarizes some of our efforts to examine long-held beliefs about common, effective data fusion techniques. Prior work demonstrates that significant

* The authors would like to note that a greatly expanded version of this work was published in the 2003 ACM Symposium on Applied Computing (ACM-SAC). Interested readers are invited to refer to [Beit03] for further details

improvement is often seen when using standard data fusion algorithms on an arbitrary collection of result sets from different information retrieval systems. This belief is supported by the supposition that different document retrieval strategies will rank documents differently, returning different sets of relevant and non-relevant documents. Several popular result combination algorithms, such as CombSUM and CombMNZ [Fox94] have been invented to take advantage of this property, using a combination of voting and merging to fuse results from several sources into one unified set. Although much research has been conducted, precise analysis of why and when data fusion techniques improve retrieval has not yet been undertaken. This may be because Lee's overlap correlation [Lee97] is generally accepted to be true. It states that fusion will generally improve effectiveness as long as the relevant overlap between component result sets is greater than non-relevant overlap. Because of this, researchers are more focused on using fusion as a utility to improve their systems than on discovering the details of what actually makes fusion a worthwhile endeavor.

We have examined the case of using data fusion techniques to fuse result sets created by different, highly effective modern retrieval strategies. A key difference between our approach and prior approaches is that we performed our experiments in a completely controlled environment; only retrieval strategy was varied across component result sets. All other systemic differences such as parsing rules, stemming rules, relevance feedback techniques, stopword lists, etc, were held constant. This approach has allowed us to examine if different retrieval strategies are actually returning different sets of documents, as generally believed. Our preliminary experiments have shown that in actuality, highly effective retrieval strategies tend to return result sets with a high degree of general overlap (ie, sets containing the same documents). In addition, we have found that Lee's overlap correlation does not hold true when highly effective strategies are used and systemic differences are held constant. When we examined the problem further, we found that in the cases where fusion is improving, it is not due to the agreement of several systems on what documents are relevant, but rather, it is due to the increase in recall of relevant documents that only appear in one component result set, and the insertion of these unique relevant documents into the final fused result set at a position of high rank.

The remainder of this paper will give a brief overview of prior work, followed by a description of our preliminary experiments and a discussion of our results. We close with a brief discussion of future work in this area, and provide references to more detailed analyses for the interested reader.

2 Prior Work

There exists a very large body of prior work in the area of data fusion. Many different types of evidence of relevance have been utilized in an attempt to improve retrieval, including different query representations, different document representations and indexing strategies, and different retrieval strategies, or methods of finding a measure of similarity between a query and a document. A variety of different techniques for utilizing this information in a data fusion strategy can be found in the literature, although the most common are Fox & Shaw's CombSUM and CombMNZ measures [Fox94]. These techniques are useful in several different applications of information retrieval, including the ad-hoc retrieval task commonly associated with the annual Text Retrieval Conference (TREC), as well as tasks in the area of distributed information retrieval and metasearch on the web.

One of the earliest studies in data fusion was performed by Belkin and colleagues [Belk93, Belk95]. They investigated the effect of fusing results from different query representations and concluded that combining multiple pieces of evidence was nearly a surefire way to increase retrieval effectiveness, suggesting that as more evidence of relevance becomes available for combination, greater improvement can be expected.

Belkin's conclusions led to further research in the area of data fusion. Lee did some initial work in trying to maximize effects gained from data fusion by exploring the effectiveness of combining the results from several term-weighting schemes with different properties in order to retrieve more types of relevant documents [Lee95]. He found that when performing combinations in this matter, significant improvements could be achieved. Lee furthered his efforts on data fusion with another study that proposed a correlation between the level of difference between relevant and non-relevant overlap among component systems and the degree of improvement that can be expected from voting/merging fusion techniques such as CombMNZ [Lee97]. Specifically, Lee stated that as long as the component systems being used for fusion had greater relevant overlap than non-relevant overlap, improvement would be observed, although an optimal ratio of these quantities was not provided. The formulae for calculating relevant overlap and non-relevant overlap for component result sets $S_1 \ldots S_n$ are shown in Equation 1 below.

$$ROverlap = \frac{R \cap S_1 \cap S_2 \ldots \cap S_n}{(R \cap S_1) \cup (R \cap S_2) \cup \ldots (R \cap S_n)} \quad (1)$$

$$NROverlap = \frac{NR \cap S_1 \cap S_2 \ldots \cap S_n}{(NR \cap S_1) \cup (NR \cap S_2) \cup \ldots (NR \cap S_n)}$$

The experimentation provided in the study shows significant improvements for fused result sets, thus appearing to support the overlap correlation. Another popular avenue for optimizing data fusion improvements gave even more weight to Lee's proposed overlap correlation. A series of studies was performed using linear combinations of sources - essentially giving a weight of confidence in the quality of a source before fusing with a common results combination algorithm like CombMNZ. Bartell and colleagues were responsible for some of the first work done in linear combinations [Bart94]. Positive results were achieved, however, the experiments were performed using a very small test collection (less than 50MB). In addition, many others have experimented in this area and observed results that seem to agree with Lee's overlap correlation.

Given that results exist which show data fusion to be effective, there is a surprising lack of detail surrounding the analysis of *why* it is effective, save for Lee's basic assumptions about overlap. To date, no detailed analysis exists in the literature of exactly how factors such as overlap and systemic differences affect the performance of fusion.

In summary, there exists a very large body of research in the area of data fusion. In spite of this, the precise reasons and conditions under which data fusion will help to improve retrieval have not been precisely specified. Lee comes closest to identifying a possible indicator for when fusion is a worthwhile approach, however, there is a lack of research exploring the specific case of fusing results from highly-effective document retrieval strategies while holding systemic differences constant. This question is what led us to examine the data fusion problem in greater detail.

3 Methodology

Our goal is to discover if retrieval strategies alone are responsible for the effectiveness improvements observed from data fusion. Furthermore, we wish to target this examination towards the fusion of modern, highly effective retrieval strategies. To analyze this problem, we must identify the cases where fusion techniques are able to provide improvements in retrieval effectiveness.

Data fusion techniques can improve retrieval in two ways. First, voting can be employed in order to boost the rank of documents that are common amongst component result sets. This point of benefit makes clear the source of Lee's statements regarding overlap. If the percentage of relevant overlap is significantly higher than the percentage of non-relevant overlap, the voting mechanisms should be more likely to boost the ranks of relevant documents, thereby improving retrieval effectiveness. However, when considering the case of highly effective retrieval strategies, we believe that voting is actually far more likely to hurt retrieval effectiveness. The reasoning for this lies in the fact that, because the component strategies are highly effective, it is fair to assume that the ranking they provide for their results is already of fairly high quality (i.e., relevant documents are likely to already be ranked higher than non-relevant documents). Given this, voting is more likely to boost a common non-relevant document to a higher rank than a common relevant document. If this occurs enough times, any improvements gained from boosting relevant documents may be cancelled out, and retrieval effectiveness may even be degraded. This leads us to establish the first part of our two-part hypothesis: when fusing highly effective retrieval strategies, the voting properties of multiple-evidence techniques such as CombMNZ will not improve effectiveness.

The second way that CombMNZ-like fusion techniques can positively affect retrieval is if they are able to merge relevant documents that are unique to a single component system into the final fused result set. This increases recall, and may increase average precision if the new relevant documents are inserted into the fused result set at high enough ranks, thereby bringing improvements to retrieval effectiveness. A caveat of this is that when the component result sets have a high degree of relevant overlap, the likelihood of merging in unique relevant documents, especially at high ranks, will tend to be very small. This leads to the second part of our hypothesis, which states that highly effective retrieval strategies tend to retrieve the same relevant documents, and therefore it is very unlikely that unique relevant documents will be merged into the final result set, and effectiveness will not be improved. When both points of our hypothesis points are taken together, the goal of this work becomes clear: if there are no improvements to effectiveness when all systemic differences are held constant and only retrieval strategies are varied, any improvements observed from data fusion *cannot* be due to the retrieval strategies.

To prove our hypothesis we designed many experiments that measure the effectiveness of both the voting and merging properties of data fusion using CombMNZ. If it can be shown that neither beneficial property of fusion is bringing improvement when holding constant all systemic differences and only varying retrieval strategies, then we have proved our hypothesis.

4 Results

For our experiments, we implemented three modern retrieval strategies that were recently shown to be highly effective in the TREC forum, one Vector-Space and two Probabilistic (IIT [Chow00], BM25 [Robe95], Self-Relevance [Kwok98]). A single information retrieval engine was then used with each of these retrieval strategies to evaluate query topics from the ad-hoc track at TREC 6, 7, and 8, and also query topics from the web track at TREC-9 and TREC-10. All of our experiments used only the title field of the TREC topics.

Our first experiments were designed to determine the validity of Lee's overlap correlation. It dictates that as long as there is a difference in relevant and non-relevant overlap, fusion will likely improve effectiveness. To examine this, we first used CombMNZ to fuse the results of each of our three highly effective retrieval strategies inside the same information retrieval system, and compared the effectiveness of this fused result set to the effectiveness of the best-performing single retrieval strategy out of the three. We illustrate this with average precision values in Table 1. Fusion of Effective Retrieval Strategies in the Same System.

Table 1. Fusion of Effective Retrieval Strategies in the Same System.

	Trec6	**Trec7**	**Trec8**	**Trec9**	**Trec10**
Best Strategy	0.1948	0.1770	0.2190	0.1847	0.1949
Fused Results	0.1911	0.1751	0.2168	0.1671	0.1935
% Imp. of Fused over Best	-1.90%	-1.07%	-1.005	-9.53%	-0.72%

We then performed a detailed overlap analysis of these results, shown in Table 2.

Table 2. Overlap Analysis for Same-System Fusion

	Trec6	Trec7	Trec8	Trec9	Trec10
Overlap	62.76%	61.14%	59.42%	61.61%	59.17%
Rel Overlap	89.52%	89.90%	90.23%	88.61%	85.88%
Nrel Overlap	72.93%	72.82%	72.03%	71.49%	68.94%
%Diff R/NR	22.75%	23.46%	25.27%	23.95%	24.57%

The second set of experiments testing the overlap correlation involves fusing the three best result sets from distinct TREC competitors for all years with title-only results available. A key difference in these experiments is that these result sets were all generated by separate information retrieval systems – they were not guaranteed to have used the same parsing rules, stemming rules, relevance feedback algorithms, etc, so it is clear that more is being varied here than simply retrieval strategy. The average precision values for improvement, and the overlap analysis are shown in Table 3 and Table 4.

Table 3. Fusion of Best-Performing TREC Systems

	Trec6	Trec7	Trec8	Trec9	Trec10
Best TREC System	0.2876	0.2614	0.3063	0.2011	0.2226
Fused Results	0.3102	0.2732	0.3152	0.2258	0.2441
% Imp. Of Fused over best	7.86%	4.51%	2.91%	12.28%	9.66%

Table 4. Overlap Analysis for Best-TREC Fusion

	Trec6	Trec7	Trec8	Trec9	Trec10
Overlap	34.43%	39.31%	42.49%	30.09%	33.75%
Rel Overlap	83.08%	80.84%	84.63%	85.85%	81.87%
NRel Overlap	53.33%	56.36%	57.13%	51.26%	54.01%
% diff R/NR	55.78%	43.44%	48.14%	67.48%	51.58%

When fusing separate systems (the TREC systems), we do see small to moderate improvements with fusion, however, if the overlap correlation were true, and if our resilts were to be consistent with those found by Lee, our effectiveness improvements should have been more substantial, and generally increased as the difference in relevant and non-relevant overlap increased. This is clearly not the case, as can be seen from Tables 1-4 above. Generally, overlap is lower (30-43% - see Table 4) in cases where there is some improvement over the best system (2.9-12.3% - see Table 3), as opposed to cases where little or no improvement (and occasionally loss) is observed (59-63% - see Table 1 and Table 2).

To further test our hypothesis, we examined our supposition that fusion only yields improvement when the component result sets contain a relatively large number of unique relevant documents. To measure this, we took each component result set and merged them such that the top X documents were examined, and any document appearing in more than one result set was discarded. This was done for various values of X so that we could observe the number of unique relevant documents present at different depths of the component result sets. The above experiments were done both for fusion of the best TREC systems and for the fusion of the three highly effective retrieval strategies in the same system. We plotted out the results in a series of graphs, one per TREC-Year. Each graph shows the percentage of uniquely relevant documents present at various depths of examination. Two curves are shown on each graph: one representing the fusion of the top three TREC systems for that year (marked as "best"), and a second curve representing the fusion of the three highly effective strategies in the same information retrieval system.

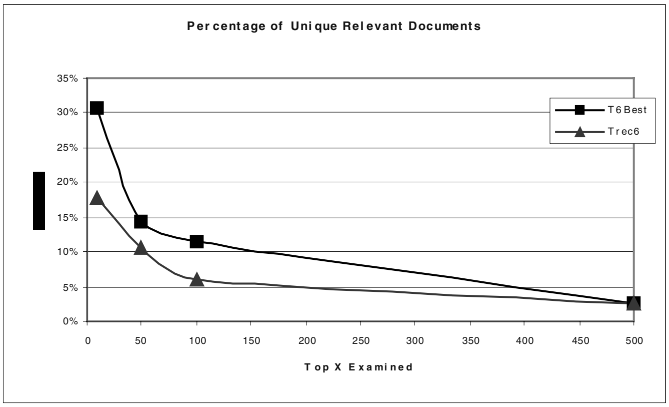


Fig. 1. TREC-6

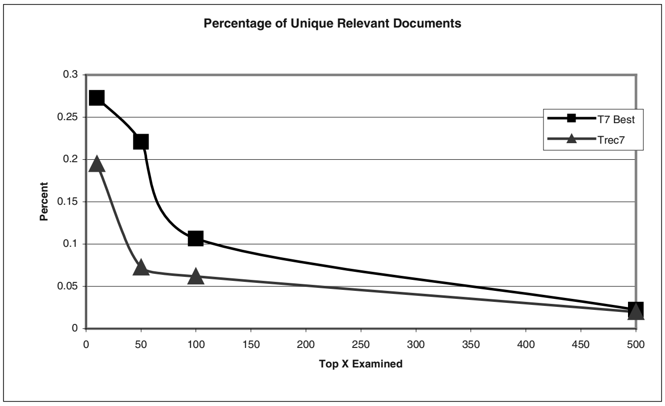


Fig. 2. TREC-7

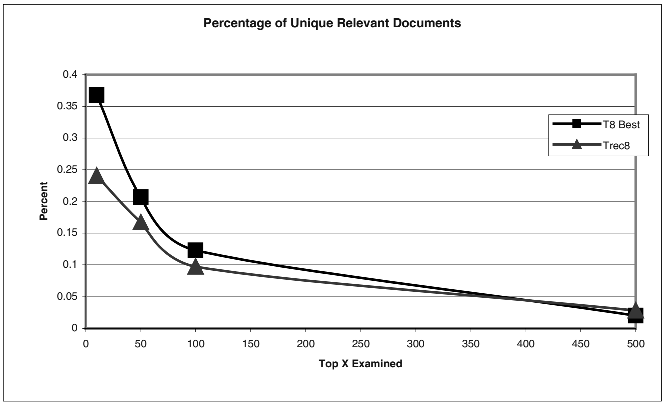


Fig. 3. TREC-8

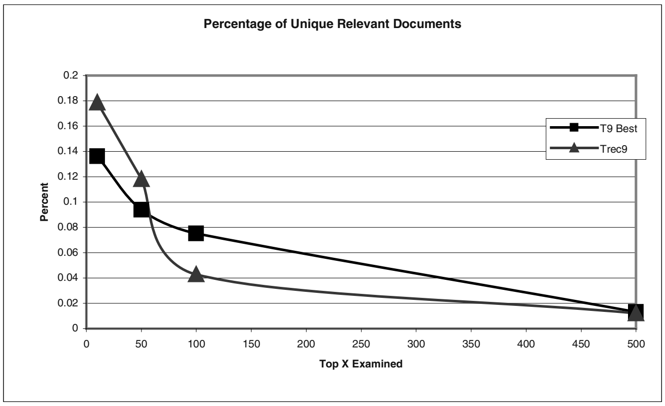


Fig. 4. TREC-9

These graphs above clearly show that for each TREC year, the fusion of the top three systems has a higher percentage of unique relevant documents in its result set for a given depth X. It is particularly interesting to note that the percentage of unique relevant documents is always greatest near the top of the result set. This means that recall is improved for the highest ranked documents. If our hypothesis about the relationship between percentage of unique relevant documents and effectiveness improvements is correct, then according to the graphs above we would expect to see that the fusion of the top 3 systems always yield a greater improvement over the best single system.

Referring back to Table 1 and Table 3 shows us that our data concurs with this expectation. To explain this we can first refer back to the earlier observation that the percentage of unique relevant documents in the result set was always at its highest when examining only the top documents in each component set. Therefore, when

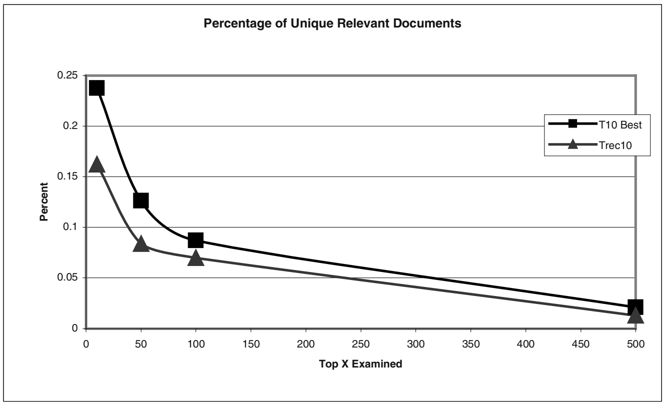


Fig. 5. TREC10

this is true, the probability of having a noticeable effect on average precision is high since fusion is allowing recall to improve by merging in different relevant documents at the highest ranked positions in the result set. Greater clarity can be achieved by examining the average number of unique (across component sets) relevant and non-relevant documents added to the result set at various depths by fusion:

Table 5. Average number of Unique Relevant and Non-Relevant documents added in same-system fusion.

Depth	R	NR	Ratio
10	0.72	3.18	0.23
50	1.29	11.83	0.11
100	1.53	21.97	0.07
500	1.60	89.84	0.02

Table 6. Average number of Unique Relevant and Non-Relevant documents added in TREC-best fusion.

Depth	R	NR	Ratio
10	1.49	4.30	0.35
50	3.46	19.77	0.17
100	3.93	36.63	0.11
500	3.19	157.61	0.02

It can be seen from the tables above than in cases where fusion shows improvement (TREC-best), the average number of relevant documents added to the highly ranked documents (depth = 10) is roughly doubled over the same-system case, while the average number of non-relevant documents is only increased by 25%.

It is still desirable to explain why multiple-evidence alone is not enough to yield significant improvement for fusion over the best single system when fusing highly effective systems or retrieval strategies. The reason for this is simply because fusing sets of documents that are very highly similar (i.e., they have high general overlap), then multiple-evidence techniques will simply scale the scores of the majority of the documents and will not help in separating relevant documents from non-relevant ones. Consequently, when general overlap is high, the number of unique (non-repeated) documents will be lower, and improvements due to fusion will be very unlikely.

5 Conclusions and Future Work

We have experimentally shown that multiple-evidence alone is not enough to ensure effectiveness improvements when fusing highly effective retrieval strategies. In order to use data fusion techniques for improving effectiveness, there must be a large percentage of unique relevant documents added to the fused set as highly ranked results, not a simple difference between relevant and non-relevant overlap as previously thought. We investigated and identified the relationship between overlap of result sets and fusion effectiveness, demonstrating that fusing result sets with high overlap are far less likely to yield a large improvement than fusing those with low overlap, if the sets being fused are highly effective. We also identified that varying systemic differences amongst result sets tends to bias improvements that have been seen in fusion experiments from the prior work, and shown that when these differences are removed, causation factors of fusion are more easily studied. For future work, we plan to investigate the specific effects that various systemic variations have on fusion effectiveness, and research the development and performance of new and existing intelligent data fusion algorithms that might overcome the limitations of those commonly used today.

References

[Bart94] B.T. Bartell, G.W. Cottrell, and R.K. Belew, “Automatic Combination of multiple ranked retrieval systems,” Proceedings of the 17th Annual ACM-SIGIR, pp. 173–181, 1994.

[Beit03] S. M. Beitzel, E. C. Jensen, A. Chowdhury, D. Grossman, O. Frieder, N. Goharian, "Disproving the Fusion Hypothesis: An Analysis of Data Fusion via Effective Information Retrieval Strategies", *Proceedings of the 2003 ACM Symposium on Applied Computing (ACM-SAC)*, Melbourne, FL, March 2003.

[Belk93] N.J. Belkin, C. Cool, W.B. Croft and J.P. Callan, “The effect of multiple query representations on information retrieval performance,” Proceedings of the 16th Annual ACM-SIGIR, pp. 339–346, 1993.

[Belk95] N.J. Belkin, P. Kantor, E.A. Fox, and J.A. Shaw, “Combining evidence of multiple query representation for information retrieval,” Information Processing & Management, Vol. 31, No. 3, pp. 431–448, 1995.

[Chow00] A. Chowdhury, et al., "Improved query precision using a unified fusion model", Proceedings of the 9th Text Retrieval Conference (TREC-9), 2000.

[Fox94] E.A. Fox and J.A. Shaw, "Combination of Multiple Searches," Proceedings of the 2nd Text Retrieval Conference (TREC-2), NIST Special Publication 500-215, pp. 243–252, 1994.

[Kwok98] K. Kwok, et al., "TREC-7 Ad-Hoc, High precision and filtering experiments using PIRCS", Proceedings of the 7th Text Retrieval Conference (TREC-7), 1998.

[Lee95] J.H. Lee, "Combining Multiple Evidence from Different Properties of Weighting Schemes," Proceedings of the 18th Annual ACM-SIGIR, pp. 180–188, 1995.

[Lee97] J.H. Lee, "Analyses of Multiple Evidence Combination," Proceedings of the 20th Annual ACM-SIGIR, pp. 267–276, 1995.

[Robe95] S. Robertson, et al., "Okapi at TREC-4", Proceedings of the 4th Text Retrieval Conference (TREC-4), 1995.

The MIND Architecture for Heterogeneous Multimedia Federated Digital Libraries

Henrik Nottelmann and Norbert Fuhr

Institute of Informatics and Interactive Systems, University of Duisburg-Essen, 47048 Duisburg, Germany, {nottelmann,fuhr}@uni-duisburg.de

Abstract. In this paper we describe the architecture of the MIND system for federating multimedia digital libraries. MIND integrates heterogeneous, multimedia non-co-operating digital libraries and gives the user the impression of a single coherent system. The architecture consists of a single mediator and one proxy (composed of several proxy components) for every connected library. These specialised, distributed components are connected via SOAP. This architecture with clearly defined responsibilities perfectly fits the needs of a distributed research project, and allows for integrating different platforms and programming languages.

1 Introduction

Today, people have routine access to a huge number of heterogeneous and distributed digital libraries. To satisfy an information need, three different steps have to be performed:

1. Relevant libraries have to be selected ("resource selection").
2. The information need has to be reformulated for every library w. r. t. its schema ("schema mapping") and query syntax.
3. The results from the selected libraries have to merged ("data fusion").

This is an ineffective manual task for which accurate tools are desirable.

MIND (being developed in an EU project) is an end-to-end solution for federated digital libraries which covers all these issues. We started from information retrieval approaches which focus on retrieval quality, but mostly only consider monomedial and homogeneous sources. We extended these approaches for dealing with different kinds of media (text, facts, images and transcripts of speech recognition) as well as handling heterogeneous libraries (e.g. with different schemas). Another innovation is that MIND also considers non-co-operating libraries which only provide the query interface.

2 The MIND Architecture

The goal of the MIND project was to develop methods and a prototype for integrating heterogeneous multimedia digital libraries (DLs). Some of these

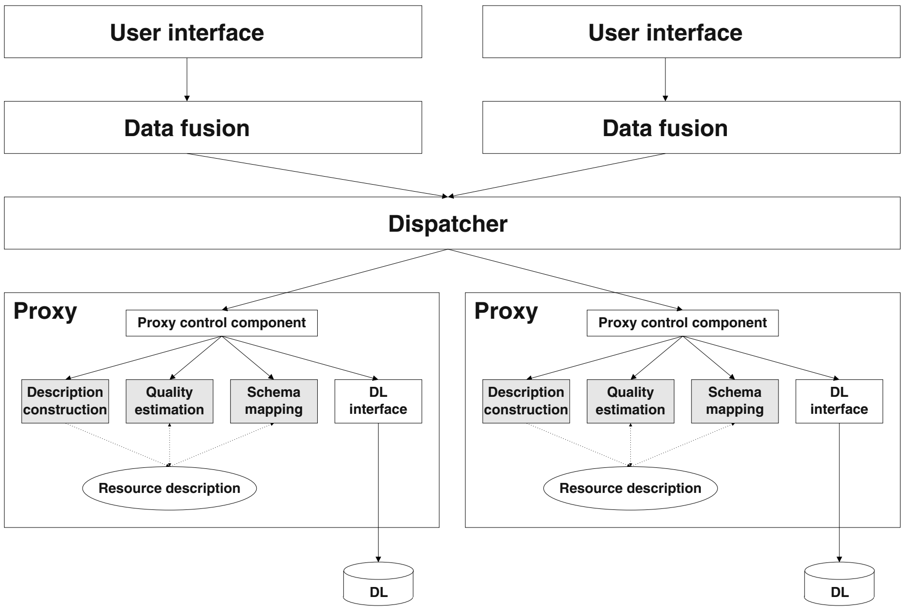


Fig. 1. MIND basic architecture

DLs might co-operate with the project, but most of them do not. Heterogeneity appears in different forms: query languages (e.g. proprietary languages, SQL, XQuery, XIRQL), communication protocols (HTTP GET/POST, Z39.50, SOAP), document models (relational, DOM) and the logical and semantic document structure (defined by schemas).

The MIND architecture reflects these three characteristics (heterogeneity, multimedia, non-co-operating DLs). It follows the standard structure with one mediator (called "dispatcher" in MIND) and wrappers ("proxies") for every library. The proxies extend the functionality of the libraries; they deal with the heterogeneity problem, and they give the dispatcher the required information not provided by the libraries. The only requirement for the DLs is that we can issue queries to it and receive documents, and that a DL client can specify the number of documents to be returned.

With this architecture, the dispatcher "only" has to deal with co-operating proxies. On top of the dispatcher, a "data fuser" merges the results together. The basic components of this architecture are depicted in Fig. 1.

The proxies consists of several sub-components (the grey components in Fig. 1 are media-specific):

Proxy control component: This component is called by the dispatcher (actually, it is the only one used by the dispatcher). The major job of this component is to call the other components, and to combine their results.

Description construction: This media-specific component is used for creating resource descriptions. Resource descriptions store textual descriptions of the corresponding library, e.g. its schema and statistical metadata (e.g. average term indexing weights).

Quality estimation: This media-specific component is responsible for estimating the number of relevant documents in the corresponding DL w. r. t. its media type. This is a sub-task of the resource selection problem (see Sect. 6).

Schema mapping: This media-specific component translates between the user schema and the DL schema: Queries have to be transformed from the user schema into the DL schema, and documents have to be transformed back from the DL schema into the user schema (see Sect. 4).

DL interface: The interface component is the connection from the proxy to the underlying library. Thus, this component provides a uniform access API.

Using external resource descriptions allows for reusing standard implementations of components (despite the DL interface, which has to be re-implemented for every proxy). Nevertheless, this approach is flexible so that different proxy implementations can be used as well.

We decided to develop a distributed system, where the components can reside on different machines. Communication is done via SOAP (see Sect. 3). This has several advantages over a monolithic architecture:

1. Components can be developed easily by the responsible partner, and integration is quite easy, without the problem of maintaining the sources and compiled programs on different sites.
2. Components can run on a dedicated computer which provides required resources (computation power, additional software or data sources like a thesaurus).
3. Different hardware, operating systems and programming languages can be used.
4. Load balancing and replication can improve the system scalability and reliability.

The disadvantage is that communication is more complex than in a homogeneous and centralised environment.

Concluding, our architecture shifts the main functionality into the "smart" proxies which extend the underlying DLs. The proxies also solve the heterogeneity problem; the dispatcher only sees a collection of homogeneous, co-operating libraries which adhere to a standard protocol (i.e., the MIND protocol).

3 Communication with SOAP

The hierarchical MIND communication structure can be implemented with simple procedure calls in a centralised environment. But as the MIND components are distributed, this solution has to be extended to remote procedure calls (RPC).

When the development of the prototype started in 2001, we decided to use SOAP[1]. In contrast to other solutions like XML-RPC[2], the W3C language SOAP now is the standard for connecting components (so called "web services").

[1] `http://www.w3.org/TR/SOAP/`

[2] `http://www.xmlrpc.com/`

SOAP messages are encoded in XML and typically transmitted via HTTP. The SOAP service has to process the request XML document and to send back a “response” XML document. The major advantage of this text-based SOAP is that it improves interoperability between different platforms and languages. In MIND, SOAP components are implemented in C++, Perl and Java.

The XML request document contains the envelope (the root element) and the body element (which contains the payload as its immediate child element). The payload (body) element is identified by a fully qualified name, where the component name is the namespace URI and the procedure name is the local name. In MIND, the component name has the form “urn:NAME”, e.g. “urn:dispatcher.Dispatcher” for the dispatcher or “urn:proxy-estimation-text.Google” for the text-specific quality estimation component of the Google proxy. Coding the service name into the XML message allows for attaching several components to one URL (i.e., to one HTTP server).

This is an example SOAP message for sending a query to the dispatcher, and returning result documents:

```
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <mindcall:query xmlns:mindcall="urn:dispatcher.Dispatcher">
        <query xmlns:mindtype="urn:MINDType"
               xsi:type="mindtype:Query">
          ...
        </query>
    </mindcall:query>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Here, `urn:dispatcher.Dispatcher-Name` is the name of the service which is called, and `query` is the procedure name. The first and only parameter is `query` (the query itself).

This is a typical response:

```
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <mindcall:queryResponse xmlns:mindcall="service urn"
        SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <return xmlns:mindtype="urn:MINDType"
              xsi:type="mindtype:Result">
        ...
      </return>
    </mindcall:queryResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

To reduce administration overhead, we wanted a central mapping from component names to URLs, the “registry”. So, only the URL of this additional SOAP service has to be stored on the client (e.g. in a configuration file). This centralised registry stores pairs like:

```
urn:dispatcher.Dispatcher=http://www.mind-project.org/soap
```

For replication, more than one URL can be specified for the same service. For improved efficiency, clients should (and do) cache the URLs retrieved from the registry, so that the client does not have to query the registry before every SOAP call. As the registry is only used for the SOAP communication, it is not included in the original architecture (see Fig. 1).

Most of the MIND code is written in Java, using the Apache SOAP implementation.[3] Here, a client calls a “router” servlet, which processes the incoming SOAP messages, converts the payload into Java objects (with specialised seriializer classes), calls the correct object (identified by the SOAP component name), and converts the result back into XML. For simplicity, we made the SOAP protocol transparent by introducing “stub” classes which implements the same interface as the component, but whose methods call the component’s method via SOAP (as in RMI). This proved to be good, as then only the Java method in the stub has to be called, and the SOAP protocol is hidden. This communication process is depicted in Fig. 2.

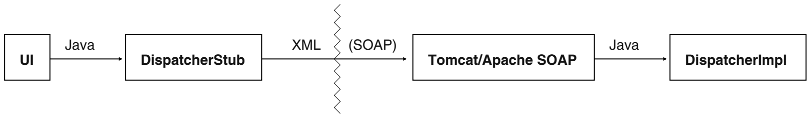


Fig. 2. MIND communication process in Java

As SOAP is an interoperability protocol, Java clients can call services written in another programming language, and vice versa.

The query-based sampling component (see Sect. 5) is C++ software (from the Lemur[4] toolkit). Within the project, it has been extended so that it can be integrated in MIND. The query-based sampling component uses the gSOAP toolkit.[5]

4 Heterogeneous Schemas

In this section, we describe and discuss the transformation of queries and documents between heterogeneous schemas.

[3] `http://xml.apache.org/soap/`

[4] `http://www-2.cs.cmu.edu/~lemur/`

[5] `http://www.cs.fsu.edu/~engelen/soap.html`

4.1 Query and Document Transformation

In heterogeneous digital libraries, each DL uses its own schema. Thus, queries have to be transformed from the user schema into the DL schema, and documents from the DL schema into the user schema.

In MIND, we employed a combined approach of DAML+OIL, probabilistic Datalog and XSLT [7]. The goal is to define schema mapping rules in a descriptive, textual way. A schema mapping rule specifies how a given attribute in one schema can be transformed into another attribute w. r. t. another schema; the attribute can be renamed, and the value can be modified. The latter is important for facts, where e.g. dates have to the transformed between different formats. Usually, text attributes are not modified (only renamed if necessary). As described in [7], MIND queries and documents are modelled in DAML+OIL [3], which has the power to become the standard ontology language.

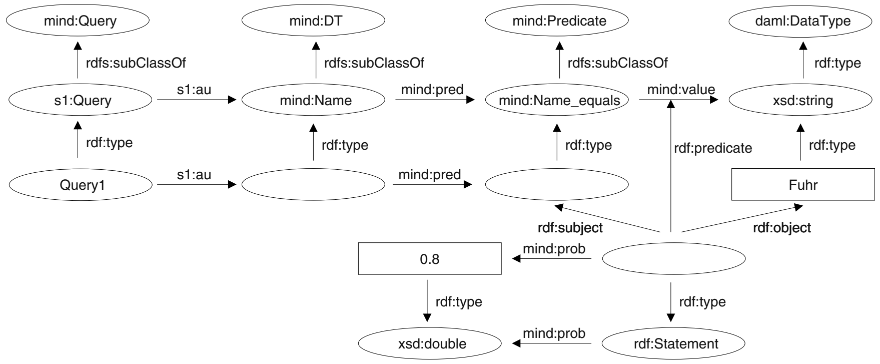


Fig. 3. Queries in DAML+OIL

DAML+OIL is an RDF extension. RDF [6] is developed in the context of the Semantic Web as a specification language for objects and their properties via statements of the form "subject predicate object" (also called triples). The subject of the statement is an RDF object, the predicate is a property, and the object of a statement is another RDF object or a literal (a value, e.g. a string or a number). The vocabulary (object classes, properties) can be defined with the schema specification language RDF Schema (RDFS for short) [1]. DAML+OIL enriches RDFS with more advanced primitives: E.g. the range of a property can be specified w.r.t. the domain (in RDFS, it can only be specified globally). As there will never be one single standard ontology, schema mapping remains a crucial issue in the DAML+OIL world.

Thus, we decided to map the MIND document model onto DAML+OIL, i.e. that MIND schemas are modelled as DAML+OIL schemas.

One example is depicted in Fig. 3. The first "level" introduces some common concepts used in the MIND document model, the second row defines the MIND schema S_1 and a concrete data type. The lower part models one specific query condition. The document attribute *au* is modelled by anonymous instances of `mind:Name`, `mind:Predicate` and `rdf:Statement` (reification).

One disadvantage of DAML+OIL is that it lacks rules so far. Thus, DAML+OIL models are converted into probabilistic Datalog [5], a variant of predicate logic based on function-free Horn clauses. Negation is allowed, but its use is limited. Rules and facts have the form

```
father(X,Y) :- parent(X,Y) & male(X).
male(peter). female(mary). 0.9 parent (peter,mary).
```

Here, `father(X,Y)` is the head of the rule, and the right part forms the body, `X` is a variable, and `peter` is a constant. So, the rule states that male parents are fathers. The facts specify that Peter is a man and Mary is a woman. The probability that peter is parent of Mary is only 0.9.

Schema mapping rules are stated in this predicate logic language:

```
s2#title(D,V) :- s1#ti(D,V).
0.7 s2#author(D,V) :- s1#au(D,V).
0.3 s2#editor(D,V) :- s1#au(D,V).

s1#ti_contains(D,V) :- s2#title_contains(D,V).
0.7 s1#au_equals(Q,V) :- s2#author_equals(Q,V) & !s2#editor_equals(Q,V).
0.3 s1#au_equals(Q,V) :- !s2#author_equals(Q,V) & s2#editor_equals(Q,V).
1.0 s1#au_equals(Q,V) :- s2#author_equals(Q,V) & s2#editor_equals(Q,V).
```

This specifies that `ti` can be mapped onto `title` and vice versa. In contrast, `au` has to be transformed into `author` with probability 0.7, and into `editor` with probability 0.3. The exclamation mark denotes negated literals.

On the implementation level, our approach is more general than evaluating pDatalog rules directly: DAML+OIL models are serialised in XML. XSLT is a well-established technology for transforming an XML document into other textual formats, e.g. another XML document (mapping between different DTDs or transforming the content), an HTML document (for creating web pages) or simple text. XSLT processors are freely available for a large number of programming languages.

The XSLT stylesheets are created based on the pDatalog rules. The transformation of pDatalog rules into XSLT is done once after the mapping rules are set up, and can be performed completely automatically. We obtain one stylesheet for transforming documents and another one for transforming queries.

For images, this text based approach does not work, so the schema mappings are implemented manually in this case.

4.2 Implementation in MIND

The proxy transforms a user query into a DL query when necessary, thus hiding heterogeneity to the dispatcher. In other words, the dispatcher never notices that the proxy internally uses another schema. In a similar way, documents are

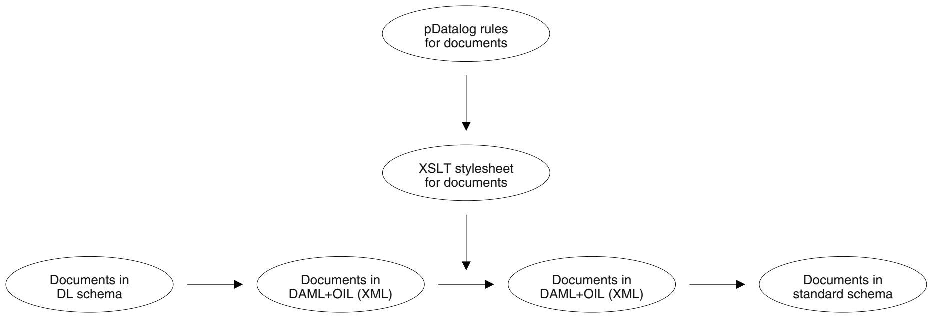

pDatalog rules for documents
XSLT stylesheet for documents
Documents in DL schema
Documents in DAML+OIL (XML)
Documents in DAML+OIL (XML)
Documents in standard schema

Fig. 4. Document transformation process

transformed automatically from the DL schema into the user schema before they are returned to the dispatcher.

The proxy splits the query (the document, respectively) into media-specific fractions. Each fraction is forwarded to the corresponding media-specific schema mapping component.

For text, facts and speech, three steps are performed (see Fig. 4):

1. The documents (internal objects, referring to the DL schema) are converted into DAML+OIL and serialised in XML.
2. The resulting XML document is transformed into another XML document by the document transformation XSLT stylesheet. The resulting XML document is also a serialisation of a DAML+OIL model, corresponding to the standard schema.
3. The resulting XML document is parsed and converted to new internal objects.

The new internal objects are returned to the proxy control component. There, the transformed fractions are recombined. For images, schema transformation with XSLT does not work, so it is performed in DL-specific program code.

4.3 Advantages and Disadvantages of This Approach

As images are handled in a way different to that of text, facts and speech, our distributed architecture with media-specific schema mapping components helps us in solving this problem.

However, we noticed that the differences for the three other media types are smaller than expected before. Thus, one schema mapping component for text, facts and speech would be sufficient and more efficient.

Our descriptive approach with probabilistic Datalog rules and the standard transformation language XSLT for processing queries and document allows for storing all library-specific information in textual files. Thus, only one standard implementation is required for all proxies.

5 Acquisition of Resource Descriptions

This section describes and discusses how metadata of a library ("resource descriptions") are acquired in MIND.

5.1 Query-Based Sampling

The second area where resource descriptions are used (besides the schema mapping task) is resource selection. For this, the resource descriptions have to contain data (e.g. average indexing weights) which can be used for estimating the retrieval quality of a DL. As MIND integrates non-co-operating digital libraries which only provide the query interface, the proxies cannot simply request resource descriptions from the DLs.

Thus, we employed the technique of query-based sampling [2]. An initial single-term query is sent to the library, and the top-ranked documents (typically four) are retrieved and added to the document sample. Then, a random term is chosen from the sample and used as the next query. This process is iterated until the sample converges, or a specified number of documents is retrieved. Finally, the sample can be used for extracting statistical metadata. So, with reasonably low costs (i.e., number of queries), an accurate resource description can be constructed from samples of, e.g., 300 documents.

Although originally developed for the CORI resource selection algorithm, it can be used for several other resource selection approaches (GlOSS, language models), including the decision-theoretic framework used in MIND.

During query-based sampling, it is easy to measure the computation and communication time of the underlying DL for that query, and to add this to the resource description. These measurements are used later for estimating the costs w. r. t. time in resource selection (see Sect. 6).

Query-based sampling is only possible for text. This means that the DL schema must contain at least one text attribute, so that this attribute can be used for sampling. Of course, the documents of the resulting sample documents can be used for deriving resource descriptions for all schema attributes.

5.2 Implementation in MIND

Query-based sampling in included in the Lemur toolkit developed by UMass and CMU and implemented in C++. For MIND, the query-based sampling part of Lemur has been extended by the project partner CMU so that it can sample MIND proxies (i.e., the query-based sampling part of Lemur can send SOAP calls to MIND proxies).

Query-based sampling is started with an external program, which issues a SOAP call to the corresponding proxy control component. This component— the entry point for all interaction with a proxy— calls a query-based-sampling component containing the Lemur code. This component then interacts with the proxy again for sampling the underlying library.

5.3 Advantages and Disadvantages of This Approach

Query-based sampling in MIND is a good example for the advantages of using a distributed environment with a platform- and language-independent data exchange format like SOAP: We were able to integrate C++ software into a Java-based environment fairly easy. We were also able to handle calls in two directions: The proxy control component (written in Java) calls the query-based sampling code (C++), which then has to call iteratively the proxies for sampling the libraries.

The disadvantage is, of course, that we have to maintain and deploy another component. Also, the sampling time increases due to a larger communication overhead, but this does not matter for query-based sampling (which is done very infrequently).

6 Resource Selection

In this section we briefly explain the MIND resource selection implementation.

6.1 Decision-Theoretic Resource Selection

MIND employs the decision-theoretic framework for resource selection [9,8]. The basic assumption is that we can assign specific retrieval expected costs (as the costs are unknown in advance) $EC_i(s_i, q)$ to each digital library DL_i when s_i documents are retrieved for query q. The term “costs” is used in a broad way and covers different sources:

Effectiveness: Probably most important, a user is interested in getting many relevant documents. Thus we assign user-specific costs C^+ for viewing a relevant document and costs $C^- > C^+$ for viewing an irrelevant document. If $E[r_i(s_i, q)]$ denotes the expected number of relevant documents in the result set when s_i documents are retrieved from library DL_i for query q, we obtain the cost function:

$$EC_i^{rel}(s_i, q) := E[r_i(s_i, q)] \cdot C^+ + [s_i - E[r_i(s_i, q)]] \cdot C^-. \tag{1}$$

Time: This includes computation time at the library and communication time for delivering the result documents over the network. These costs can easily be approximated by measuring the response time for several queries. In most cases, a simple affine linear cost function is sufficient:

$$EC_i^{time}(s_i, q) := C_0^{time} + C_1^{time} \cdot s_i. \tag{2}$$

Money: If a DL charges for its usage, this has to be specified manually. In MIND, we only connect DLs which do not charge. However, if a DL charges, in most cases a purely linear (reflecting per-document-charges) is appropriate: per-document basis.

$$EC_i^{money}(s_i, q) := C^{money} \cdot s_i. \tag{3}$$

Thus, in most cases we have $C^{money} = 0$.

A user can specify the importance of the different cost sources by user-defined cost parameters. Thus, a user can specify her own selection policy (e.g. cheap and fast results with a potentially smaller number of relevant documents). The expected costs $EC_i(s_i, q)$ then is the weighted sum of the costs of the different sources.

If the user specifies (together with her query) the total number n of documents which should be retrieved, the task then is to compute an optimum solution, i.e. a vector $\boldsymbol{s} = (s_1, s_2, \ldots, s_m)^T$ with $|\boldsymbol{s}| = \sum_{i=1}^{m} s_i = n$ which minimises the overall costs.

The optimum selection $\boldsymbol{s}$ can be computed by the algorithm presented in [4].

6.2 Implementation in MIND

In MIND, three components are involved in resource selection: the quality estimation component in the proxy, the proxy control component and the dispatcher.

Corresponding to the MIND philosophy of proxies extending non-cooperating DLs, costs $EC(s, q)$ are estimated by the proxy. The optimum selection—based on the costs of every proxy—can only be computed by the dispatcher.

For time and monetary costs, the MIND model uses simple approximations. These costs can be computed in the media-independent proxy control component. The situation is more complex for relevancy costs: Here, the probabilities of relevance $Pr(\mathrm{rel}|q, d)$ have to be estimated.

As this estimation is data-type- and media-specific, it is delegated to the media-specific quality estimation components. Thus, the query (already transformed from the user schema into the DL schema) is split into media-specific sub-queries. Each sub-query q_m is forwarded to the corresponding quality estimation component; the result is a list of n probabilities of relevance $Pr(\mathrm{rel}|q_m, d)$ of the n top-ranked documents (assumed that the user requested n documents) w. r. t. the sub-query. Of course, each estimator can choose its own method for deriving these probabilities, but all of them require data from the resource descriptions (e.g. average indexing weights).

For text, we approximate the indexing term weights by a normal distribution, and use these approximations for estimating the number of relevant documents in the result. For the other media types, we simulate retrieval on the resource description (a document sample as a subset of the underlying DL), and extrapolate the results on the whole collection.[6] Details can be found in [9,8].

After all, they are combined in the proxy control component: the probabilities of relevance of the first entries in each line are combined, then the probabilities of relevances of the second entries, and so on (see Fig. 5).

6.3 Advantages and Disadvantages of This Approach

The MIND approach has a clear and natural separation of responsibilities: the media-specific component estimates the probabilities of relevance w. r. t. media-

[6] For text, this second method performs worse than the normal distribution one.

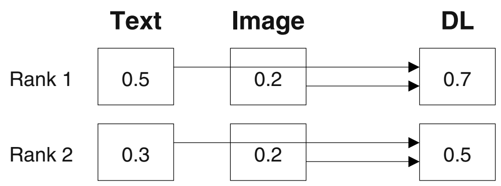


Fig. 5. Combination of probabilities of relevance from the different media-specific estimations

specific sub-queries, the proxy control component combines the probabilities and computes costs for the different cost sources, and the dispatcher computes—based on the cost estimation from all the proxies—an optimum selection.

In this decentralised approach of estimating retrieval quality, it is very easy to deal with heterogeneous schemas: Each quality estimation component only has to deal with one DL, and only sees queries in the DL schema. As the resource descriptions (and, thus, the statistical metadata required for resource selection) are stored on the proxy level, they are also separated from resource descriptions of other DLs, so heterogeneity of schemas is not a problem at all.

However, this MIND approach also has two disadvantages:

1. Each proxy has to be called (if it runs on another machine than the dispatcher, this means a SOAP call over the network), and each proxy has to query its own resource description. In the current MIND implementation, most of the resource description data is stored in a relational database management system, as this makes it easier and more efficient when dealing with large data. However, this means that every proxy has to issue at least one SQL command to the database. Thus, resource selection in MIND is quite expensive.
2. The probabilities of relevance of the sub-queries have to be combined in the proxy control component. As each list only contains n documents, we combine the first probabilities, then the second and so on. However, the probabilities in the first ranks do not necessarily belong to the same (the first) document. For a precise combination, the probabilities of relevance and document ids for all documents in the sample are required; but this would lead to high communication costs in the proxy.

7 Alternative Architectures

Obviously, the MIND architecture is a compromise between various requirements of different types. Five other architectures could be considered as well:

1. A monolithic, centralised architecture without distributed components is faster (as there is no communication time), but does not allow for replication. Furthermore, it does not fit the needs of a distributed research project where each partner works on separate research problems.

2. The MIND architecture could live with less components. E.g., the description construction component could be fused with the quality estimation component, as they are strongly related. In addition, the text-, fact- and speech-specific schema mappings components work exactly the same and thus could be fused. Quality estimation for text differs from estimation for facts. However, for an increased estimation quality, estimating and communicating probabilities of relevance for all documents in the DL would be useful. Thus, one could also fuse these three media-specific quality estimation components and deal with the differences inside this component.
3. Resource selection could be much faster if the costs were estimated in the dispatcher. Then, the relational database used in MIND could be queried only once, and there would be no need for calling each associated proxy. Both would lead to faster resource selection. The disadvantage is that it is more difficult to deal with heterogeneous schemas: one would either transform the sample documents to the standard schema and then create the resource description (which makes it difficult and expensive to change the standard schema), or the resource descriptions for all attributes have to be stored in one huge database table (which would decrease processing time inside the database).
4. Communication in MIND is quite expensive due to the XML-based SOAP protocol. The client has to encode the internal objects into XML, the XML record (which is much larger than the internal representation) has to be transmitted over the network, and the service component has to parse the XML structure. Other native protocols on a binary basis (e.g. CORBA) would be more efficient, but it more difficult to achieve interoperability between different platforms and programming languages. In addition, CORBA cannot be operated over a firewall, which was used in one of the project partner's department. Another solution would be that the components operate directly on the SOAP XML tree, without converting it to internal objects. However, this would lead to a more expensive programming work, which is not suitable for a prototype system which is developed in parallel with scientific research.
5. Nowadays, peer-to-peer systems become popular. A peer-to-peer architecture is more flexible than the MIND communication structure, removes control from any single organisation, and makes it easier for new service providers to join the federation. We will deal with this kind of architecture in a forthcoming project.

8 Conclusion and Outlook

In this paper we described the MIND architecture for heterogeneous multimedia federated digital libraries. MIND follows a highly distributed approach with components for every distinct task. Communication is done via SOAP, which easily allows for integrating different platforms and programming languages. This architecture perfectly fits the needs for a distributed project where each partner works on a different part of the overall research problem, and where development is done in parallel to scientific research.

In a latter project stage, it turned out that some components could be fused together, but time constraints did not allow for doing so. Also, the overall system slows down due to communication time. Thus, our architecture is a little bit oversized in the MIND context. In a production system, the system could easily be adapted.

For similar projects, however, our architecture has the big advantage that is flexible and extensible. So, new functionality can be added quite easily and in a natural way, either in one of the existing components or in new ones.

In later versions of the system, we will downsize the architecture to the level actually required. In a new project, we will extend our approach to peer-to-peer nets.

Acknowledgements. This work is supported by the European Community under IST Project MIND (IST-2000-26061).

References

[1] D. Brickley and R. Guha. RDF vocabulary description language 1.0: RDF Schema, w3c working draft. Technical report, World Wide Web Consortium, Apr. 2002.

[2] J. Callan and M. Connell. Query-based sampling of text databases. *ACM Transactions on Information Systems*, 19(2):97–130, 2001.

[3] D. Connolly, F. v. Harmelen, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. DAML+OIL (march 2001) reference description. Technical report, World Wide Web Consortium, 2001. `http://www.w3.org/TR/daml+oil-reference`.

[4] N. Fuhr. A decision-theoretic approach to database selection in networked IR. *ACM Transactions on Information Systems*, 17(3):229–249, 1999.

[5] N. Fuhr. Probabilistic Datalog: Implementing logical information retrieval for advanced applications. *Journal of the American Society for Information Science*, 51(2):95–110, 2000.

[6] O. Lassila and R. R. Swick. Resource description framework (RDF) model and syntax specification. W3C recommendation, World Wide Web Consortium, Feb. 1999. `http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/`.

[7] H. Nottelmann and N. Fuhr. Combining DAML+OIL, XSLT and probabilistic logics for uncertain schema mappings in MIND. In *European Conference on Digital Libraries (ECDL 2003)*, Heidelberg et el., 2003. Springer.

[8] H. Nottelmann and N. Fuhr. Decision-theoretic resource selection for different data types in MIND. In *Proceedings ACM SIGIR 2003 Workshop on Distributed Information Retrieval*, New York, 2003. ACM.

[9] H. Nottelmann and N. Fuhr. Evaluating different methods of estimating retrieval quality for resource selection. In *Proceedings of the 26st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, New York, 2003. ACM.

Apoidea: A Decentralized Peer-to-Peer Architecture for Crawling the World Wide Web

Aameek Singh, Mudhakar Srivatsa, Ling Liu, and Todd Miller

College of Computing,
Georgia Institute of Technology,
Atlanta, GA - 30332
{aameek,mudhakar,lingliu,tomiller}@cc.gatech.edu

Abstract. This paper describes a decentralized peer-to-peer model for building a Web crawler. Most of the current systems use a centralized client-server model, in which the crawl is done by one or more tightly coupled machines, but the distribution of the crawling jobs and the collection of crawled results are managed in a centralized system using a centralized URL repository. Centralized solutions are known to have problems like link congestion, being a single point of failure, and expensive administration. It requires both horizontal and vertical scalability solutions to manage Network File Systems (NFS) and load balancing DNS and HTTP requests.
In this paper, we present an architecture of a completely distributed and decentralized Peer-to-Peer (P2P) crawler called Apoidea, which is self-managing and uses geographical proximity of the web resources to the peers for a better and faster crawl. We use Distributed Hash Table (DHT) based protocols to perform the critical URL-duplicate and content-duplicate tests.

1 Introduction

Search engine technology has played a very important role in the growth of the WWW. Ability to reach desired content amidst huge amounts of data has made businesses more efficient and productive. An important component of this technology is the process of *crawling*. It refers to the process of traversing the WWW by following hyperlinks and storing downloaded pages. Most of the currently available web crawling systems [5,10] have envisioned the system as being run by a single organization. The architecture of such a system is based on a central server model, which determines which URLs to crawl next and which web pages to store and which to dump (in case, a similar page is already available in the database by maybe, crawling a mirror site).

Such a system requires organizations to employ extremely resourceful machines and various experienced administrators to manage the process. For example, Mercator [10] used a machine with 2 GB of RAM and 118 GB of local disk. Google [5] also utilizes a special hardware configuration and centralized management of tens and thousands of machines to crawl the web. Along with higher costs, an important problem arises because of the congestion of the link joining the crawling processes with the central server, where all the data is kept.

With the emergence of successful applications like Gnutella [9], Kazaa [11], and Freenet [7], peer-to-peer technology has received significant visibility over the past few years. Peer-to-peer (P2P) systems are massively distributed computing systems in which peers (nodes) communicate directly with one another to distribute tasks or exchange information or accomplish tasks. P2P computing has a number of attractive properties. First, it requires no additional hardware, and the computing resources grow with clients. Second, it is self-organizing, incurring no additional administrative costs due to scaling. Third but not the least, it is fault-tolerant by design. P2P systems differ from each other in terms of how they exchange data or distribute tasks, and how they locate information (lookup service) across the overlay network.

In this paper we present a peer to peer approach to crawl the web. Apoidea[1] is a fully decentralized P2P system with no central authority, much like the family of bees it derives its name from. All the bees (peers) work independently without any commanding authority to decide further crawling and the data is kept distributed across the network. The task of crawling is divided amongst the peers and we use DHT based distributed lookup and information-exchange protocols to exchange vital information between the peers. Also, we use bloom filters [3] to store the list of URLs already crawled by a peer. This makes the memory requirements at each peer very reasonable and easily available at normal PCs of today.

The option of distributing the task of crawling across the internet also provides us of an opportunity to exploit geographical proximity of crawlers to the domains they are crawling. Our initial results indicate that using such a feature can significantly speed up the crawling process. Also having a decentralized system makes it possible to crawl the web with minimal or no supervision and using well studied replication mechanisms, such a design is inherently more fault tolerant.

Please note that in this paper, we focus only on the crawling of the WWW. We do not present any mechanisms for indexing and providing a complete search engine. These mechanisms have been well studied and can be used on top of our design architecture.

The main advantages of Apoidea are as follows:

– **Efficient**: Apoidea utilizes peers geographically closer to the web resources to crawl. This can lead to significant speed improvements and greater efficiency. It also serves as an automatic load balancing mechanism for DNS servers and local proxy servers.
– **Fully Decentralized**: Apoidea is completely decentralized. This prevents any link congestion that might occur because of the communications with a central server. The peers perform their jobs independently and exchange information whenever they need to.

[1] Apoidea is a family of bees which does not have any queen bee

- **Low Cost**: Apoidea is a self-managing system. It means there is no need of manual administration. The system automatically handles the dynamics of P2P systems - the entry and exit of peers. Also the resource requirements at every peer are very reasonable.
- **Greater Reach**: With the increase in the number of peers in the system, we can potentially reach a much greater part of the WWW as opposed to what conventional systems can do.
- **Scalability**: Apoidea has been constructed with an aim of scalability in mind and we envision the system to work efficiently with huge WWW growth rates.
- **Fault Tolerance**: Since Apoidea takes into consideration entry and exit of peers, the system, by design, is more fault tolerant than the ones available today, which have a single point of failure.

The rest of the paper is organized as follows:
Section-2 describes the basics of crawler design and a look at DHT based distributed lookup protocols and bloom filters. *Section-3* describes the complete architecture and working details of Apoidea. In *Section-4*, we present our initial results and observations of the performance of the system. In *Section-5*, we discuss how Apoidea could be used to build a World Wide Web search engine. In *Section-6*, we discuss the work related to this paper. Finally, *Section-7*, contains our conclusions and future directions of research.

2 System Design: The Basics

In this section, we briefly describe general design requirements for a web crawler. Then we will talk about the work done in the area of DHT based P2P systems and bloom filters and how we intend to use them in our architecture. Later, in Section-3 we concretely explain how the whole system is put in place and its workflow details.

2.1 Design Considerations

In this section, we describe the basic design components of any crawler and specifically the important design principles that have to be taken into consideration for distributed and decentralized crawlers.

A typical web crawler consists of three primary components:

- *Downloader*: This component reads a list of URLs and makes HTTP requests to get those web pages.
- *URL Extractor*: This component is responsible for extracting URLs from a downloaded web page. It then puts the URL in the to-be-crawled list.
- *Duplicate Tester*: This component is responsible for checking for URL and content duplicates.

For the crawl, it also maintains the following data structures:

- *Crawl-Jobs*: It is the list of URLs to be crawled.
- *Seen-URLs*: It is a list of URLs that have already been crawled.

– *Seen-Content*: It is a list of fingerprints (hash signature/checksum) of pages that have been crawled.

The downloader picks up URLs from the Crawl-Jobs data structure and downloads the page. It is checked for a content duplication. If it turns out to be a duplicate, it is thrown away. If it is a new page, its fingerprint is added to the Seen-Content data structure. The URL Extractor component then processes this page to extract more URLs and normalizes them. The URLs are then checked for possible duplicates and if found new, are added to the Crawl-Jobs list. Along with it, they are also added to the Seen-URLs list. Both the duplicate tests have been proved to be important. Encountering a duplicate URL is extremely common, due to popularity of content and a website having common headers and footers for all its web pages. Duplicate pages occur due to mirror sites and also when there are symbolic links at a web server, for example, both home.html and index.html point to the same page on many web servers. In Mercator's [10] experiments, around 8.5% of the pages were duplicates.

For a distributed crawler, decisions have to made for selecting which crawler gets to crawl a particular URL. This can be as simple as just picking up URLs from Crawl-Jobs sequentially and giving it to the crawlers in the round robin fashion or can be based on a complicated policy. For example, Google [5] uses various machines in a round robin manner and the Internet Archive [6] crawler distributes them based on domains being crawled.

However, designing a decentralized crawler has many new challenges.

1. **Division of Labor**: This issue is much more important in a decentralized crawler than its centralized counterpart. We would like the distributed crawlers to crawl distinct portions of the web at all times. Also there are potential optimizations based on geographical distribution of the crawlers across the globe. In Section-3, we describe how we exploit the global presence of the peers in the Apoidea system.

2. **Duplicate Checking**: Even assuming each peer crawling a distinct part of the web, they will still encounter duplicate URLs and duplicate content. Since there is no single repository of Seen-URLs and Seen-Content data structures, the peers have to communicate between each other and decide on the newness of a URL or a page. And it is extremely important to keep these communication costs to a minimum since we can potentially have thousands of peers across the globe crawling the web at a very fast speed. We use DHT based protocols, which achieves these lookups in $O(log\ n)$ time, where n is the total number of peers.

It is important to notice that the above two components are not entirely disconnected from each other. Having a good division of labor scheme can potentially save us on the duplicate decision making part.

2.2 Crawling the Web Using a Peer to Peer Network

In the recent past, most of the research on P2P systems was targeted at improving the performance of search. This led to the emergence of a class of P2P systems that include Chord [17], CAN [13], Pastry [14] and Tapestry [2]. These techniques are fundamentally based on distributed hash tables, but slightly differ in algorithmic and implementation details. All of them store the mapping between a particular *key* and its *value* is a distributed manner across the network, rather than storing them at a single location like a conventional hash table. This is achieved by maintaining a small routing table at each node. Further more, given a *key*, these techniques guarantee the location of its *value* in a bounded number of hops within the network. To achieve this, each peer is given an identifier and is made responsible for a certain set of keys. This assignment is typically done by normalizing the key and the peer identifier to a common space (like hashing them using the same hash function) and having policies like numerical closeness or contiguous regions between two node identifiers, to identify the regions which each peer will be responsible for.

For example, in the context of a file sharing application, the key can be a file name and the identifier can be the IP address of the peer. All the available peers' IP addresses are hashed using a hash function and each of them store a small routing table (for example, for Chord the routing table has only m entries for an m bit hash function) to locate other peers. Now, to locate a particular file, its filename is hashed using the same hash function and depending upon the policy, the peer responsible for that file is obtained. This operation of locating the appropriate peer is called a *lookup*.

A typical DHT-based P2P system will provide the following guarantees:

- A lookup operation for any key k is guaranteed to succeed if and only if the data corresponding to key k is present in the system.
- A lookup operation is guaranteed to terminate within a small and finite number of hops.
- The key identifier space is uniformly (statistically) divided among all currently active peers.
- The system is capable of handling dynamic peer joins and leaves.

Apoidea uses a protocol similar to that of Chord [17], because of its simplicity in implementation. In Apoidea, we have two kinds of keys – URLs and Page Content. The term – *a peer is responsible for a particular URL*, means that the URL has to be crawled by that particular peer and for the page content, it means that the peer has information whether that page content has already been downloaded (probably by crawling a mirror site) or not.

Now, a URL lookup can be done as follows. First, we hash the **domain name** of the URL. The protocol performs a lookup on this value and returns the IP address of the peer responsible for this URL. Note that this choice will result in a single domain being crawled by only one peer in the system. It was designed in this manner, since now a peer can use the *Connection-Alive* parameter of the HTTP protocol and use a single TCP/IP socket while downloading multiple pages from a particular domain. This significantly fastens the crawl, because we save on the costs of establishing a TCP/IP connection with the domain, for every URL crawled, which would have been the case if we had just hashed the complete URL and done a lookup on it.

The page-content is used as a key when we intend to check for duplicate downloaded pages. For this, the page-content is hashed and a lookup is initiated to get the peer responsible for this page-content. Note that, the page could (and in most case, would) have been downloaded by another peer (since that is based on the lookup of the URL's domain name). The peer responsible for this page content only maintains the information and does not store the entire page.

Also, because of the P2P nature, the protocol must handle dynamic peer join and leave operations. Whenever a new peer joins the system, it is assigned responsibility for a certain portion of the key space. In the context of a web crawler, the new peer is assigned those domains that hash onto it. The peers that were currently responsible for those domains are required to send information about the list of URLs already crawled in those domains to the newly entering node. A very similar technique is used to redistribute the page signature information.

When a Peer P wants to leave the network, it sends all URL and page content information currently assigned to it to other nodes. However, if the Peer P were to fail, then it would not be able to send any information to other peers; that is the information about the keys held by Peer P would be temporarily lost. This problem is circumvented by replicating information. Every domain D is mapped to multiple keys $(k_1, k_2, \cdots, k_R)$ which are in turn handled by distinct peers. However only the peer responsible for key k_1 (*primary replica*) is responsible for crawling the domain D. The peers responsible for the secondary replicas (keys $k_2, k_3, \cdots, k_R$) would simply store information as to which URLs in domain D have been already crawled or not. However, if the peer responsible for the primary replica crashes, the information held by it could be recovered using one of its secondary replicas. For the sake of simplicity, we will consider only having one key for each domain in the rest of the paper.

2.3 Managing Duplicate URLs Using Bloom Filters

Bloom filters provide an elegant technique for representing a set of key elements $K = \{k_1, k_2, \cdots, k_n\}$ to support membership queries. The main idea is to allocate a vector of m bits, initially set to 0 and choose l independent hash functions $h_1, h_2, \cdots, h_l$ with range $\{0, 1, \cdots, m-1\}$. For any key $k \in K$, the bit positions $h_1(k), h_2(k), \cdots, h_l(k)$ is set to one. Note that a particular bit location may be set to one by more than one key. Now, a membership check is performed as follows. Given a key k, we check for bits at positions $h_1(k), h_2(k), \cdots, h_l(k)$ in vector m. If any of them is zero, then certainly $k \notin K$. Otherwise, we assume that the given key $k \in K$ although our conclusion could be wrong with a certain probability. This is called a *false positive* .

The salient feature of the Bloom Filters is in the fact that one can trade off space (m) with the probability of a false positive. It has been shown in [3] that using $l = ln2 * m/n$ independent hash functions, one can reduce the probability of false positive to $(0.6185)^{\frac{m}{n}}$. We can achieve a probability of false positive as low as 0.001 for $l = 5$ independent hash functions and $m/n = 10$.

In the context of a web crawler, we can use a bloom filter to maintain the information about a URL having been crawled or not. All the URLs that have already been crawled form the set K. Now, to test a URL for its newness, we hash it to a value k and then check for membership of k in K. This will indicate whether that URL has already been crawled or not. Assuming that the WWW contains about 4 billion pages and setting $m/n = 10$ the size of our bloom filter would be $4 * 2^{30} * 10$ bits or 5GB. Note that any practical implementation of the bloom filters would like to maintain the entire filter in its main memory. Hence, it is infeasible to hold the entire bloom filter required for the scale of the WWW in a single machine. In our design we divide the URLs in WWW amongst many cooperating peers. Assuming that we have only a 1000 cooperating peers in different geographical locations then each of these peers would have to handle about 4 million web pages, which would in turn require a bloom filter of size 5MB. Given today's technology, each peer can easily afford to maintain this 5MB bloom filter in its main memory.

3 System Architecture

Apoidea consists of a number of peers all across the WWW. As explained in the previous section, each peer is responsible for a certain portion of the web. In this section we first describe how we perform the division of labor and how the duplicate checking is done in Apoidea. Then we briefly describe the peer-to-peer (P2P) architecture of Apoidea, especially the membership formation and the network formation of peers, and the single peer architecture in Apoidea, including how each peer participates in the Apoidea web crawling and performs the assigned crawling jobs.

– **Division of Labor:** As we have already discussed in Section-2, we use a DHT-based system for distributing the WWW space among all peers in the network. A Peer P is responsible for all URLs whose domain name hashes onto it. In Apoidea, we do it using the contiguous region policy, which will be clearer from the example shown in Figure-1. Assume that there are three peers in the system - Peer A, B and C. Their IPs are hashed onto an m bit space and hence, will be located at three points in the modulo m ring. Then, various domain names are also hashed onto this space and they will also occupy such slots. The Peer A is made responsible for the space between Peer C and itself. Peer B is responsible for space between Peer A and itself and Peer C with the rest. From the figure, Peer A is responsible for the domain *www.cc.gatech.edu*. Therefore Peer A will be the only peer crawling all the URLs in that domain. If any other peer gets URLs belonging to that domain, it batches them and periodically send them to Peer A.
  However, such a random assignment of URLs to peers does not exploit the geographical proximity information. In order to ensure that a domain

is crawled by a peer *closer* to the domain, we relax the tight mapping of a domain to a peer as follows. Every Peer P maintains a list of *leaf peers* that are close to Peer P in the identifier space. In the context of the Chord protocol, a Peer P maintains pointers to $l/2$ successor peers and $l/2$ predecessor peers on its identifier ring. Now, when a Peer P is assigned to domain D, it picks the leaf peer closest to domain D and forwards the batch of URLs to it for crawling.

– **Duplicate Checking:** Let us assume that Peer P is responsible for domain D and that Peer Q a leaf peer of Peer P is actually crawling domain D. When any peer encounters a URL in domain D, it sends the URL to Peer P since it is responsible for domain D. Now, Peer P batches a collection of URLs in domain D and forwards them to Peer Q. Recall that Peer Q maintains a bloom filter which indicates as to whether a URL is already crawled or not. Hence, Peer Q checks if a given URL is already crawled or not. If the URL were indeed already crawled then it is simply dropped, else it is added to the Crawl-Jobs list.
  In order to check page duplication, we hash the page contents onto the identifier space and hence distribute them amongst the various peers in the network. Figure-2 shows an example of it. The page content of *www.cc.gatech.edu/research* is hashed onto the ring and Peer C is responsible for it. Note that the page would have been crawled by Peer A because the domain *www.cc.gatech.edu* hashed onto it. When Peer A encounters that page, it needs to check whether that page is a duplicate or not. So it looks up for the hash of page content and finds out Peer C is responsible for it. Now Peer A can batch all such requests and periodically query Peer C about the newness. Peer C can then batch the replies back. It also modifies its local Seen-Content list to note the new pages that have been downloaded by Peer A. Note that the delay in getting the information about the duplicity of a page is not performance-effecting since always, there is a significant delay in downloading a particular page and processing that page. This is because the downloads typically happen at a much faster rate than the processing and as a result the processing has a significant lag from the download.

3.1 Apoidea Peer-to-Peer Network Formation

Peers in the Apoidea system are user machines on the Internet that execute web crawling applications. Peers act both as clients and servers in terms of their roles in performing crawling jobs. An URL crawling job can be posted from any peer in the system. There is no scheduling node in the system and neither does any peer have a global knowledge about the topology of the system. There are three main mechanisms that make up the Apoidea peer to peer (P2P) system.

1. **Peer Membership:** The first mechanism is the overlay network membership. Peer membership allows peers to communicate directly with one

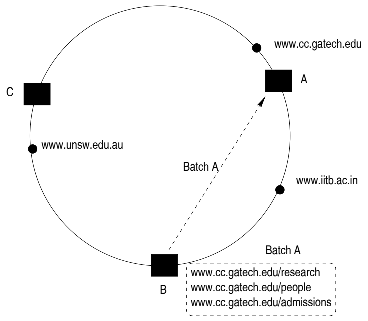


Fig. 1. Division of Labor

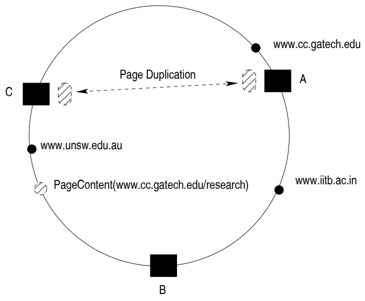


Fig. 2. Content-duplicate checking

another to distribute tasks or exchange information. A new node can join the Apoidea system by contacting an existing peer (an entry node) in the Apoidea network. There are several bootstrapping methods to determine an entry node. We may assume that the Apoidea service has an associated DNS domain name. It takes care of resolving the mapping of Apoidea's domain name to the IP address of one or more Apoidea bootstrapping nodes. A bootstrapping node maintains a short list of Apoidea nodes that are currently alive in the system. To join Apoidea P2P Web crawling, a new node looks up the Apoidea domain name in DNS to obtain a bootstrapping node's IP address. The bootstrapping node randomly chooses several entry nodes from the short list of nodes and supplies their IP addresses. Upon contacting an entry node of Apoidea, the new node is integrated into the system through the Apoidea protocol's initialization procedures. We will revisit the process of entry and exit of nodes in Section-3.3.

2. **Protocol:** The second mechanism is the Apoidea protocol, including the partitioning of the web crawling jobs and the lookup algorithm. In Apoidea every peer participates in the process of crawling the Web, and any peer can post a new URL to be crawled. When a new URL is encountered by a peer, this peer first determines which peer will be responsible for crawling this URL. This is achieved through a lookup operation provided by the Apoidea protocol.

   The Apoidea protocol is in many ways similar to Chord [17]. It specifies three important types of peer coordination: (1) how to find peers that are best to crawl the given URL, (2) how new nodes join the system, and (3) how Apoidea manages failures or departures of existing nodes. A unique feature of the protocol is its ability to provide a fast distributed computation of a hash function, mapping URL crawl jobs to nodes responsible for them.
3. **Crawling:** The third mechanism is the processing of URL crawling requests. Each URL crawling job is assigned to a peer with an identifier matching the domain of the URL. Based on an identifier matching criteria, URLs are crawled at their assigned peers and cleanly migrated to other peers in the presence of failure or peer entrance and departure.

From a user's point of view, each peer in the P2P network is equipped with the Apoidea middleware, a two-layer software system. The lower layer is the Apoidea P2P protocol layer responsible for peer-to-peer communication. The upper layer is the web crawling subsystem responsible for crawling assigned URLs, resolving last-seen URLs to avoid duplicate crawling, and processing crawled pages. Any application-specific crawling requirements can be incorporated at this layer.

In the subsequent sections we discuss the Apoidea middleware that runs at every Apoidea peer to describe the design of the P2P crawling subsystem in more detail. We also report some initial experimental results obtained by evaluation of the first prototype of Apoidea P2P Web crawlers.

3.2 Single Peer Architecture

The architecture of every peer is composed of three main units:

1. **Storage**: This unit contains all the data structures that each peer has to maintain.
2. **Workers**: This unit consists of modules fetching pages from the WWW and processing them to extract more URLs.
3. **Interface**: This unit forms the interface of each peer, handling communications within the Apoidea network.

The complete architecture is shown in Figure-3.

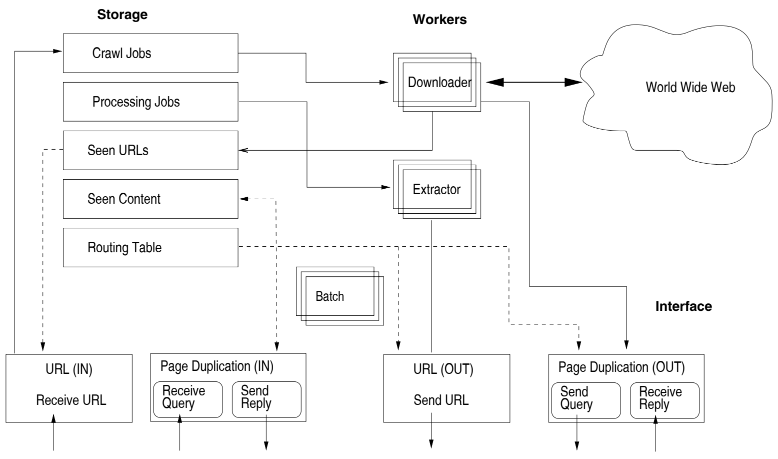


Fig. 3. System Architecture for a Single Apoidea Peer

Storage: This unit contains all the data structures maintained within each peer. These are:

– *Crawl-Jobs*: This is a list of URLs which are yet to be crawled by the peer. Since we batch URLs to a peer domain-wise, we maintain it as a hashtable with domains as the keys and the URLs belonging to that domain as the corresponding values.
– *Processing-Jobs*: This is a list of downloaded pages which are yet to be processed. The list is just a Vector of pointers to the pages. After a certain number of pages, the rest are stored on disk.
– *Seen-URLs*: This is a list of URLs that have already been crawled. This is used to prevent a same URL being crawled again. We maintain it as a bloom filter.
– *Seen-Content*: This is a list of page signatures (hash), which have already been seen by some peer in the network. An important point to be noted about this data structure of the peer is that it contains the page signatures for which this peer is *responsible for* and not the pages it downloads. Again, we use a bloom filter for this data structure.
– *Routing Table*: This is used to route the lookup queries to the right peer and it contains information about a small number of nodes in the system [17].

The biggest complexity in this component is the choice of data structures, since the web crawler is very extensive on resources, especially memory. The Crawl-Jobs and Processing-Jobs lists are usually easily maintainable since there are a number of downloaders and extractors working on them. Also the Routing-Table is a very small data structure. The main requirement is to efficiently manage the Seen-URLs and Seen-Content data structures. These data structures increase in size with the passage of time.

In order to maintain these data structures, we use bloom filters. Assume that the WWW has about 4 billion web pages. From our experiments we observed that each domain has about 4K URLs. Hence, the number of domains in the WWW would be about 1 million. Now, assuming that 1K peers participate in the crawling operation, each peer would crawl about 1K domains. For each of the domains we maintain a bloom filter that indicates whether a particular URL in that domain is already crawled or not. Noting the fact that these bloom filters need to hold about 4K URLs, we chose the size of the bloom filter to be 8KB so that the probability of false positive remains below one in a thousand (Using the formula in Section-2 and m = 8KB = 64K bits = 16 * 4K URLs). Every peer maintains a bloom filter of size 8MB (8KB per domain * 1K domains per peer). The peers also maintain the association between the domain name and its bloom filter in a hash table. Observing the fact that the average length of an URL is 80B and the hash table needs to hold 1K entries (1K domains) the size of this hash table would be about 80KB. Maintaining per domain bloom filters is essential for handling peer joins and leaves. Recall that every peer is assigned some domains. Hence, a newly joined node can construct its bloom filter for the domains assigned to it by simply copying the relevant bloom filter from the peer that was previously responsible for that domain.

Note that by using a bloom filter we have specifically avoided storing all the seen URLs in the memory. In fact, Mercator [10] uses this approach of storing all the URLs in the domain it is currently crawling in its main memory. However, Mercator reports a hit rate of only 75%, which warrants several costly disk I/O operations. Such costly disk operations might be suitable when the peer is entirely dedicated to crawling. However, in a P2P scenario one would like to minimize those operations that can significantly affect the normal functioning of the peer. Also using a peer to crawl just a single domain at one time is inefficient since having a problem with that domain (say, shut down for maintenance) will significantly reduce the throughput of the system.

Workers: This unit mainly consists two sets of threads - *Downloader* and *Extractor*.

1. **Downloader:** The Downloader threads pick up a domain from the Crawl-Jobs list and queries l neighbors maintained in its neighbor list to find out the peer which can crawl that domain fastest. This can be done in very short message exchanges with each peer measuring the round trip time to that domain. If this peer is not the fastest peer, it can just pass on the URLs to the fastest neighbor. Usually this happens because of the geographical proximity of the neighbor to the domain being crawled. As our initial experiments indicate, this can be a very important factor in the total speed of crawling the whole WWW.
   The Robot Exclusion Protocol is taken into consideration before crawling any particular website, so that any site which does not wish to be crawled is not crawled. As mentioned before, each domain is assigned to a single peer. This significantly speeds up the crawl since that peer uses the *Connection-Alive* feature of the HTTP protocol, in which a single open socket can be used to download a number of pages.
   After a page is downloaded, it is stored in a temporary data structure and checked for page duplication. If this is a new page, it is inserted into the Processing-Jobs list, else dropped.

2. **Extractor:** The Extractor threads pick up pages from Processing-Jobs list and extracts URLs from that particular page. The URLs are extracted using the regular expression library of Java. After the extraction, the URLs are normalized (changing relative URLs to exact URLs, removing extraneous extensions like "javascript:", which cannot be handled by the crawler etc). These URLs are then kept in a temporary data structure from where they are periodically picked and batched to be submitted to the appropriate peers, that is, the peers who are responsible for them. This temporary data structure is a hash table with URLs belonging to the same domain in the same bucket.

Interface: While crawling, the peers have to communicate with each other in the following scenarios:

1. *Submission of URLs*: Peer A may extract URLs (from a page it crawled) which Peer B is responsible for, that is, the domain of those URLs is hashed

to be belonging to Peer B. Peer A batches these URLs and periodically send the URLs to Peer B. Then Peer-B simply ignores the URLs that is has already crawled else it adds them to its Crawl-Jobs list.
2. *Checking for Page Duplication*: After Peer A downloads a page, it has to check if a similar page is already available in the network or not. It does a lookup on the page hash and contacts the peer responsible for that hash value, say Peer B. In case Peer-B has the same page, Peer A will drop that page, else it stores it. Also Peer B notes that such a page is available in the network now.
3. *Protocol Requirements*: Such communications are required to maintain and stabilize the network. It includes periodic message exchange between neighboring peers to identify if a new peer has joined the network or some peer has exited the network.

We use low-cost UDP messages for the exchange of this information and a set of batching threads that periodically pick up batches of information for various peers and send the data.

3.3 Entry and Exit of Peers

In this section, we will illustrate how Apoidea handles dynamics associated with a P2P system - the entry and exit of peers.

Entry: Like any P2P based system, there will be a few bootstrap servers; at all times, there will be at least one which will be up and active. These servers act as entry points into the network. Whenever a peer wants to join the Apoidea network, it will contact a bootstrap server, which will point it to a node active in the network. Depending upon the IP address of the entering peer, the node will help it get the routing table required to participate in the network. Also the network stabilizes recognizing the entry of a new node. This procedure is inexpensive and can be easily accomplished using known methods provided by Chord [17].

During this stabilization process, the node which holds a list of Seen-URLs that the new peer is responsible for now, will send that data to the new peer. This data is just a short bloom filter for the domains that hash onto the new peer. Similar transfer will take place for the page content. After a peer has joined the network, it will periodically begin to get URLs of the domains it is responsible for and it will start the crawling process.

Exit: When a Peer P needs to leave the system, its Seen-URLs and Seen-Content have to be transferred to the peer which will be responsible for it now. The new responsible peer will always be a neighbor of Peer P on the ring space. This neighbor information is always stored at the peer as part of the routing table. Therefore, before its exit, it can just send the Seen-URLs and Seen-Content information to that neighbor.

In case of a peer failure, the network stabilization process corrects the routing tables of the peers in the network. However, we would have lost the information regarding Seen-URLs and Seen-Content. To prevent this, as mentioned in Section-2, we can maintain multiple replicas of this information. The meagre requirements of memory due to the use of bloom filters makes it possible for peers to act as primary as well as secondary replicas for various keys.

4 Results and Observations

We conducted experiments on geographical proximity and the scalability of Apoidea. Using a primitive Java implementation of the crawler, we crawled four similar domains (in terms of their networking infrastructure) from Georgia Institute of Technology: *.edu* domains starting form *www.cmu.edu* in USA, *.ac.jp* domains starting from *www.u-tokyo.ac.jp* in Japan, *.edu.au* domains starting from *www.unsw.edu.au* in Australia and *.ac.in* domains starting from *www.iitb.ac.in* in India. The results from our experiment are shown in Figure-4. Clearly, the figure shows that *.edu* domains being geographically the closest can be crawled at about twice the speed of the geographically farther domains like *.edu.au*. Given the fact that the networking infrastructure of these domains are comparable, a domain like *www.unsw.edu.au* can be crawled at about twice the speed from a geographically closer location. This, in turn, can have a huge impact on the total speed of the web crawl.

We performed two experiments on scalability. In our first experiment, we measured the resource requirements (CPU + Memory) on a single peer. We ran Apoidea on a Intel Pentium 550MHz machine under two scenarios: high load (running a ns2 simulator) and low load. Note that we intentionally chose a heavily loaded machine so as to restrict the resource utilization of Apoidea. Figure-5 shows the performance of Apoidea in terms of the number of URLs crawled per second with varying number of Java threads. Observe that when the machine was otherwise heavily loaded, Apoidea's performance does not increase with the number of threads for more than two threads (other than the JVM's overhead in handling more threads), since at any time most of the threads stay in the sleep state. Also note that under high load, Apoidea's CPU utilization was restricted to 3% and its memory consumption was limited to only 30MB. However, under lightly loaded conditions, the maximum rate is achieved when a peer employs six threads. Note that under lightly loaded conditions, Apoidea's CPU utilization was 30% and its memory consumption was 30MB.

In our second experiment on Apoidea's scalability, we increased the number of peers participating in the system. We experimented on a maximum of 16 machines on the same local area network in Georgia Institute of Technology. Note that the peers were either heavily loaded or lightly loaded as in described in our previous experiment. Also, each peer used two threads under heavy load and six threads under light load so as to obtain the best performance (from Figure-5). Figure-6 shows the performance of Apoidea with varying number of peers. Note that when only one peer was running Apoidea there is no cost of interaction

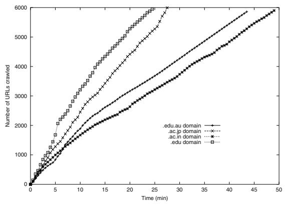


Fig. 4. Impact of Geographical Proximity

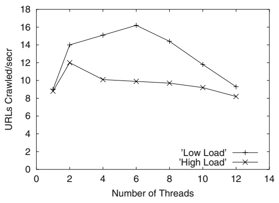


Fig. 5. Apoidea's Performance on one peer

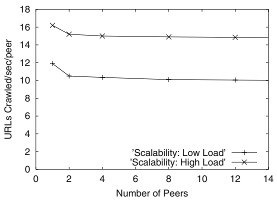


Fig. 6. Scalability of Apoidea with the Number of Peers

between the peers. Hence, adding the first peer lowers Apoidea's performance. However, note that with subsequent addition of peers, the performance almost flattens out even for small peer population of the order of 12 or 16 irrespective of the load on the peers.

In the near future, we propose to study the performance of Apoidea in the presence of a large peer population, possibly across multiple geographical locations. This will require volunteers from all over the world to run Apoidea. Second, we would like to measure the crawl refreshing speed; that is, having crawled the entire WWW once, we want to measure the speed at which the WWW can be crawled only for updates during the second and subsequent crawls. Third, we want to perform experiments on the reliability of the system and see the impact dynamic peer failures and replication can have on the overall network.

5 Application: A World Wide Web Search Engine

Several papers have proposed architectures to build scalable distributed indexes for keyword search over P2P networks [12,8]. Their designs are layered over distributed hash table based P2P systems like Chord [17]. [12], inspired by the ranking system used by Google, exploits statistics about popular documents and hosts in building distributed inverted indexes and answers a query by computing a join over the indexes matching the query keywords. On the other hand, [8] builds a distributed inverted index keyed on all combinations of multiple keywords. Thus [8] avoids the overheads of computing joins thereby saving on the network bandwidth at the cost of storage space (Note that the total storage space required grows exponentially with the set size of keyword combination used to build the inverted index).

Apoidea in conjunction with these distributed indexing architectures can be used to implement a search engine for the World Wide Web. We envision a distributed search engine wherein both Apoidea and distributed indexing architectures like [12,8] co-exist on any node in the P2P network. Apoidea crawls the World Wide Web and extracts relevant keywords from web pages. The indexing architecture builds a distributed inverted index and evaluates keyword-based search queries. Note that both Apoidea and the distributed indexing system can share a common underlying lookup protocol layer based on DHT-based P2P systems. Further, one can package the Apoidea and the distributed indexing scheme as a screen saver (like SETI@home [15]) so that the search engine on each node uses its idle CPU cycles and network bandwidth.

6 Related Work

Most of the research in the area of web crawlers has been based on centralized architectures. While [5,10,6] were a single machine architectures, [16] presents the design of a distributed crawler. The common feature of all those architectures being a centralized location storing all the critical data structures and a central manager controlling various crawler components. Such an architecture is complicated and requires high maintenance. Since the controlling organization will be using a single link with the internet, it requires a very high bandwidth connection. In addition, it requires complicated load balancing for HTTP and DNS requests.

The P2P decentralized solutions, in addition to not suffering from above mentioned issues has other potential benefits as well. Having an autonomous solution prevents costly administration requirements and is more fault tolerant. Also it has the immense potential to exploit geographical proximity of various peers to the web resources. There have been a few attempts at developing P2P decentralized versions of crawlers. Boldi et al [4] looked at the fault tolerance properties of such crawlers, showing that such an architecture will be more fault tolerant. [18] also attempted to provide a loosely connected P2P design. However, both of those works did not provide a concrete mechanism of assigning jobs to various peers and load balancing between them.

We would also like to mention the work done in the area of distributed P2P lookup protocols. There are several P2P protocols proposed so far [2,13,14,17]. Apoidea P2P web crawler is built on top of Chord [17]. Most of the routed query based P2P protocols are motivated by file sharing applications. However, none of them are really suitable for file sharing applications due to the problems in providing flexible mappings from file name search phrases to file identifiers. (One such insufficient mapping is provided in [1]). It is interesting to see that web crawling can benefit from a P2P approach, being an extremely suitable application domain for routed query based P2P protocols.

7 Conclusions and Future Work

In this paper, we have presented a decentralized P2P architecture for a web crawler. We have described the complete design and structure of the system. We show how optimizations based on geographical proximity of peers to the web resources can significantly better the crawling performance and results. We have also presented a sample mapping scheme that achieves this goal. Also, through our initial experiments we have shown the nice scalability of our architecture.

For the future, we would like to run the system in a wide scenario and test its properties. On the research side, we would like to enhance the URL mapping scheme in a way that implicitly handles geographical proximity. Also, security is a key problem when using an autonomous collection of peers. We intend to explore issues preventing malicious behavior in the network.

References

1. K. Aberer. P-grid: A self-organizing access structure for p2p information systems. In *Sixth International Conference on Cooperative Information Systems*, 2001.
2. J. Kubaitowics B. Zhao and A. Joseph. Tapestry: An infrastructure for fault-tolerance wide-area location and routing. Technical Report UCB/CSD-01-1141, University of California, Berkeley, 2001.
3. Burton Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, July 1970.
4. P. Boldi, B. Codenotti, M. Santini, and S. Vigna. Ubicrawler: Scalability and fault-tolerance issues. In *Poster Proceedings of 11th International World Wide Web Conference.*, 2002.
5. Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.
6. M. Burner. Crawling towards eternity: Building an archive of the world wide web. In *Web Techniques*, 1997.
7. Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science*, 2009:46+, 2001.
8. Omprakash D. Gnawali. A keyword-set search system for peer-to-peer networks.
9. Gnutella. The gnutella home page. http://gnutella.wego.com/, 2002.
10. Allan Heydon and Marc Najork. Mercator: A scalable, extensible web crawler. *World Wide Web*, 2(4):219–229, 1999.
11. Kazaa. The kazaa home page. http://www.kazaa.com/, 2002.
12. T. Lu, S. Sinha, and A. Sudam. Panache: A scalable distributed index for keyword search. Technical report, 2002.
13. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. 2001.
14. A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, November 2001.
15. SETI@home. The seti@home home page. http://setiathome.ssl.berkeley.edu.
16. Vladislav Shkapenyuk and Torsten Suel. Design and implementation of a high-performance distributed web crawler. In *Proceedings of International Conference on Data Engineering*, 2002.
17. I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of SIGCOMM Annual Conference on Data Communication*, August 2001.
18. T. Takahashi, H. Soonsang, K. Taura, and A. Tonezawa. World wide web crawler. In *Poster Proceedings of 11th International World Wide Web Conference.*, 2002.

Towards Virtual Knowledge Communities in Peer-to-Peer Networks

Melanie Gnasa, Sascha Alda, Jasmin Grigull, and Armin B. Cremers

Institute of Computer Science III, University of Bonn, Römerstrasse 164, 53117 Bonn, Germany
{gnasa, alda, grigull}@cs.uni-bonn.de

Abstract. As a result of the anonymity in todays Web search, it is not possible to receive a personalized search result. Neither prior search results nor search results from other users are taken into consideration. In order to resolve this anonymity towards the search engine, a system is created which locally stores the search results in the scope of a peer-to-peer network. Using the Peer Search Memory (PeerSy) all relevant results are stored and associated with the corresponding queries. By this means, repeated access is facilitated. Furthermore, sharing of associations between queries and relevant results in the peer-to-peer network allows grouping of Virtual Knowledge Communities (VKC) in order to obtain a surplus value in knowledge sharing on the Web.

1 Introduction

The rapid development of technologies associated with the Web offers the possibility of a new personalized search in a distributed environment. Today, Web search engines have to deal with two main problems: (1) no personalized search results based on preceding queries of the user exist and (2) regardless of the information need of other users these results are not taken into account as feedback information. In this light, we present a personalized approach for building Virtual Knowledge Communities (VKCs) based on personal search memories.

These VKCs represent a variety of topics, which are interesting for a multiplicity of searches. The interest is measured by the frequency of queries to a special topic. The Google search statistics Zeitgeist[1] reveals that different topics are requested more frequently in certain weeks. In addition the major search interests diverge throughout different countries. For example, the Google Zeitgeist statistics 2002 summarizes that the Google search traffic follows the Las Ketchup craze as it circles the globe. Due to the fact that today usually no personalized search is performed it is not possible to profit from the results of previous queries to the same topic. The search engine Google processes 250 million queries daily. However, these queries are not made available to the public. The benefit of these queries would only be perceivable if additional feedback information for relevant hits had been logged. So far, each user stores high-quality links locally in his

[1] http://www.google.com/press/zeitgeist.html

bookmark or favorite list [Jones et al., 2001]. Thereby an association between queries and relevant results is not provided. For the surplus value of these associations throughout the multiplicity of searches two possibilities do exist: (1) the search engine provider could save the relevant hits next to all queries in a central storage or (2) each user could save his or her relevant results locally according to the requested queries, which in turn could be shared in a distributed environment. The advantage of a pure central system is the high availability and the multiplicity of data that can be collected and retrieved. However, these systems exhibit some drawbacks, such as the existence of a single-point-of-failure: if the server fails, the whole systems will become unavailable for all depending clients. Furthermore, no support is usually provided to create VKCs to subdivide clients according to their interests or specific knowledge.

Recent peer-to-peer architectures [Barkai, 2002] are conceived to support decentralize systems as described in the second possibility. As far as reasonable, these architectures abandon from having any kind of central servers to store data. The data remains at the edges of the Internet (personal computers), thus, eliminating the single-point-of-failure problem. Each peer is thereby autonomous in the decision, to what time and to what extend data is to be shared with the environment. Another promise of peer-to-peer is the facility to build so-called peer groups, which allows groups of peers to restrict the access to distinct data to authorized peers only. In the course of our research, we propose the adoption of the peer-to-peer ideology to realize distributed knowledge communities that accomplish their members to share knowledge of Web content in terms of enriched links to specific topics.

The rest of this paper is organized as follows. Some basic assumptions about distributed knowledge sharing are illustrated in chapter 2. Chapter 3 presents PeerSy, our tool to save query-result associations in a Peer Search Memory. Our approach for Virtual Knowledge Communities is elucidated in Chapter 4. Chapter 5 gives a short survey on related work to our research. A conclusion finally sums up the main aspects of this paper and presents the future work.

2 Distributed Knowledge Sharing

In order to assist a user with his daily usage of the Web we need more than simple file sharing via search engines or peer-to-peer networks like Gnutella and Freenet. For the sharing of knowledge common Web technologies have to be adapted. This is the reason why we propose two techniques which are integrated into the knowledge creation cycle. These techniques are motivated by the transformations between implicit and explicit knowledge and the current state of the art of searching the Web.

The human knowledge can be divided into two categories: implicit and explicit knowledge (cf. [Nonaka, 1991], [Stanoevska-Slabeva et al., 1998]). The implicit knowledge depends on a person and is embedded into a certain context. In contrast, explicit knowledge is the externalisation of implicit knowledge, i.e. information. Explicit knowledge is perceived when information, context and experiences are combined to form new implicit knowledge [Nonaka, 1991]. The creation and classification of documents is part of the externalisation process. The retrieval is part of the internalisation process. The general advantage of *full text search* in the Web in the context of knowledge sharing is achieved through the automatic indexing accomplished by robots. The automatic classification makes this part of the externalisation very efficient. The general disadvantage of the common full text search engines is the lack of effectiveness of the search results (e.g. caused by synonymy and polysemy of query terms). For this reason this approach is very weak for the internalisation process in the knowledge cycle (cf. [Stanoevska-Slabeva et al., 1998]).

Ideally a system should fulfill both requirements. The full text search is on the one side highly effective in the externalisation of knowledge, on the other side, however, highly ineffective in the internalisation. In order to close this gap, we have developed two approaches for a distributed knowledge sharing:

1. **Peer Search Memory (PeerSy)**
   Each user produces individual needs for his own image of the Web through evaluated results. Thus, the problem of polysemy that occurred in the context of search engines reduces itself to a local result set in a specified context. Once an ambiguous query is requested, the anonymity against the search machine can be averted and the user receives an answer, which is filtered for his or her own context. Through the additional storage of associations between queries and results lexicographic indexing can be replaced and a grouping of synonymous terms is possible. This way search histories form the basis for effective internalisation and externalisation.
2. **Virtual Knowledge Communities (VKC)**
   The structure of a distributed retrieval environment in the Web reveals the possibility to form Virtual Knowledge Communities on the basis of search memories. Each participant can thus profit from the knowledge of the group. In contrast to the classical full text search the process of the internalisation becomes more effective.

Both techniques can be integrated into the cycle of knowledge generation. Figure 1 depicts this process. Through PeerSy implicit knowledge can be transferred into explicit knowledge. A distributed Information Retrieval System based on Virtual Knowledge Communities makes the internalisation more effective.

3 PeerSy – Peer Search Memory

PeerSy is developed to overcome the weaknesses of the conventional search machines and the current Web browsing technology with the access to well-known Web sites. PeerSy is integrated into the system ISKODOR which is being presently developed at the University of Bonn. ISKODOR is a knowledge tool for the World Wide Web on the basis of peer-to-peer technologies. With the help of the search interface MySearch, which is part of the ISKODOR system, the user can look for new or well-known Web sites. With MySearch the

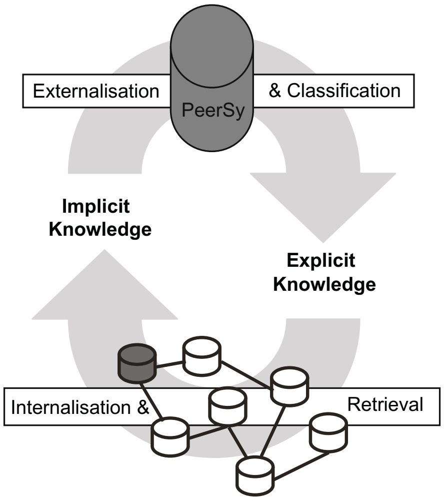


Fig. 1. Knowledge generation cycle

searching process is extended to not only take place in the Web, but also in the local PeerSy database and the PeerSy contents of other peers. All sites, which have been evaluated and rated interesting by the user are stored in the personal database PeerSy. The user this way assembles his "own personal Web", which on the one hand consists only of a tiny fraction of the entire Web and on the other hand merely contains sites, which are interesting to the user. The search for well-known sites takes place in the small cutout of the Web, in his personal database only. All associations between queries and relevant results contained in PeerSy can formally be defined as follows:

Definition 1 (Single Associations) *A set of links L which represent relevant results and a number of n query terms T for a peer p are given. A query consists of at least one query term. The set Q of all possible queries over all terms $t \in T$ is*

$$Q_p = P(T) \setminus \{\}$$

The set B of all possible relevant links to a query is

$$B_p = P(L) \setminus \{\}$$

The relation $SingleA$ between the sets Q and B describes all possible ***single associations*** *between a query and a set of relevant results, which are stored in PeerSy for peer p:*

$$SingleA_p(Q_p, B_p) = \{(q, b) \in Q_p \times B_p | (q, b) \in PeerSy_p\}$$

Thus, for each peer the individual information need is identified. A further summary of these search interests on the local peer level is waived. It cannot be assumed that certain topics are investigated representatively. Only the summary of all query-link associations over all peers makes it possible to form meaningful emphases. How these knowledge communities are formed and how the retrieval is realized to enhance the internalisation process, is described in the next section.

4 Virtual Knowledge Communities

As already brought up in the introducing chapter, peer-to-peer architectures serve as the best candidate to build Virtual Knowledge Communities. Peers span a decentral virtual network over some existing physical networks, thus, hiding the complexity of connecting peers despite firewalls or subnets (see figure 2). The ability of peers to self-organize into peer groups can be utilized to build private Virtual Knowledge Communities on top of these peers. Communities represent a place for peers sharing interests or competencies within a common knowledge domain. Appropriate authorization routines thereby have to prevent the non-restrictive access to knowledge or to resources. Note that each peer can necessarily belong to multiple groups. All these concepts have been conceptualized for instance in JXTA ([Sun, 2003]), the de-facto standard protocol suite to build peer-to-peer architectures.

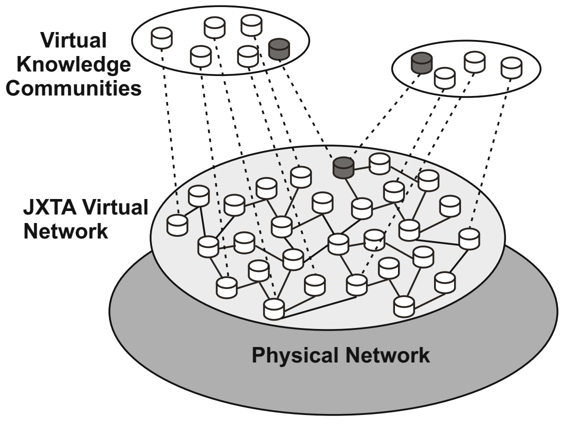


Fig. 2. Peer-to-Peer Network with semantic layer for Virtual Knowledge Communities

4.1 Building of Virtual Knowledge Communities

Similar to the clustering of data and documents [Jain and Dubes, 1988] a Virtual Knowledge Community (VKC) is described through a selected representative. The clustering of documents results in a partition of documents, which can be described by a representative. For the building of a VKC this procedure is reversed and first the possible representative of a group is computed. All peers, which have single associations that overlap with the representative can be member of this group. The growth of such Virtual Knowledge Communities is described in detail in the next subsection. The creation of representatives of VKCs from a set of n peers takes place in four steps.

In **step 1** all single associations from all peers p are unified. Thus the multi-set A contains all single associations from all peers:

$$A(Q, B) = \bigcup_{p \in P} SingleA_p(Q_p, B_p)$$

with $Q = \bigcup_{p \in P} Q_p$ and $B = \bigcup_{p \in P} B_p$. The multi-set characteristic is very important in this step in order to measure the frequency of queries and the relevant results. For example it has arisen the following multi-set A_{exp}:

$$\begin{aligned} A_{exp} = \{ \; & (\{java\}, \{java.sun.com\}), (\{jvm\}, \{java.sun.com\}), \\ & (\{java\}, \{java.sun.com, www.javaworld.com, java.apache.org\}), \\ & (\{haskell, tutorial\}, \{www.haskell.org/tutorial\}), \\ & (\{java\}, \{www.sun.com, java.sun.com\}), \\ & (\{php, tutorial\}, \{www.php.net/tut.php\}), \\ & (\{java, tutorial\}, \{java.sun.com/doks/tutorials\})\} \end{aligned}$$

In **step 2** all overlapping queries and links are unified. Through the combination of all local peer interests two kinds of overlaps can be identified between elements of A. Through the combination of all local PeerSy contents it is now possible that individual query terms emerge in different queries. Therefore it is important to form the set of overlapping queries as large as possible. The combination of all these query sets is denoted *QCloud*. For example an element from *QCloud* would be the set of queries $\{\{java\}, \{java\}, \{java\}, \{java, tutorial\}\}$ for the example set A_{exp}. Still the multi-set property of A is important and necessary to reflect repeated queries from the peers. The function *QCloud* over A describes the set of all overlapping queries $u \in U = P(Q) \setminus \{\}$:

$$\begin{aligned} QCloud(A) = \{u \in U | \bigcap u \neq \emptyset, u \subseteq domA, \\ \forall v, v \subseteq domA, \bigcap v \neq \emptyset, u \subseteq v \Rightarrow u = v\} \end{aligned}$$

In contrast to the union over the query terms, identical links can occur in the tuples $(q, b) \in A$ with $q \in Q$ and $b \in B$. With the function *LCloud* the set of all grouped links can be described:

$$LCloud[u] = \bigcup Links[u]$$

The function Links[u] is an auxiliary function and supplies the union of the pertinent link associations from the relation A to each element $u \in U$ with:

$$\begin{aligned} & Links[u] = \bigcup A[u] \\ & A[u] \text{ refers to the second position in the tuple } (q, b) \in A \\ & A[u] = \{b | \exists q \in u : qAb\} \end{aligned}$$

For example an element from *LCloud* would be the set of results

$$\{java.sun.com, java.sun.com, java.sun.com, java.sun.com,\\ www.sun.com, www.javaworld.com, java.apache.org\}$$

for the example set A_{exp}.

In **step 3** all overlapping query and link sets are summarized. By means of the relation SAQ' (Seldom Asked Queries) all queries with the corresponding links are summed up.

$$SAQ'(u, b) = LCloud|_{QCloud(A)} = \{(u, b)|u \in QCloud(A) \wedge b = LCloud[u]\}$$

It can occur that the same links are stored to different queries. For this reason all queries for these links must be summarized in SAQ. This case appears, if the same information need with different queries is described.

$$SAQ(u, b) = \{(u, b)|b \in rangeSAQ' \wedge u = \bigcup SAQ'^{-1}[b]\}$$

For the example set A_{exp} the following SAQ set can be computed:

$$SAQ_{exp} = \Big\{ (\{\{java\}, \{java\}, \{java\}, \{java, tutorial\}, \{jvm\}\},\\ \{java.sun.com, java.sun.com, java.sun.com, java.sun.com,\\ www.sun.com, www.javaworld.com, java.apache.org\}),\\ (\{\{java, tutorial\}, \{haskell, tutorial\}, \{php, tutorial\}\},\\ \{www.haskell.org/tutorial, java.sun.com/doks/tutorial,\\ www.php.net/tut.php\}) \Big\}$$

Step 4 consists of the assignment of all VKC representatives. In the last step all elements from the relation SAQ are selected that possess a certain size of associated queries and links. This size depends on x, the minimum number of queries, and y, the minimum number of links. Both values again depend on the total number of peers. At present the experimental investigation of these values takes place.

$$VKC(U, B) = \{(u, b)|\ u \in domSAQ, b \in rangeSAQ,\\ \|u\| > x \wedge \|b\| > y\}$$

After the four steps, a set of VKC representatives is found. For instance the following representative can be selected from SAQ_{exp}:

$$VKC_{exp} = \Big\{ (\{\{java\}, \{java\}, \{java\}, \{java, tutorial\}, \{jvm\}\},\\ \{java.sun.com, java.sun.com, java.sun.com, java.sun.com,\\ www.sun.com, www.javaworld.com, java.apache.org\}) \Big\}$$

The problem of different verbalizations of the same information need on individual peers can be intercepted by viewing the entire peer-to-peer net. Over

QClouds alternative queries can be reflected and over *LClouds* frequently and good-evaluated links are collected. Both functions detect synonymous descriptions of a special issue. The problem of polysemy cannot be solved by this procedure so far. However, in the future a detailed analysis of each VKC on a semantic level is planed in order to achieve a further partition.

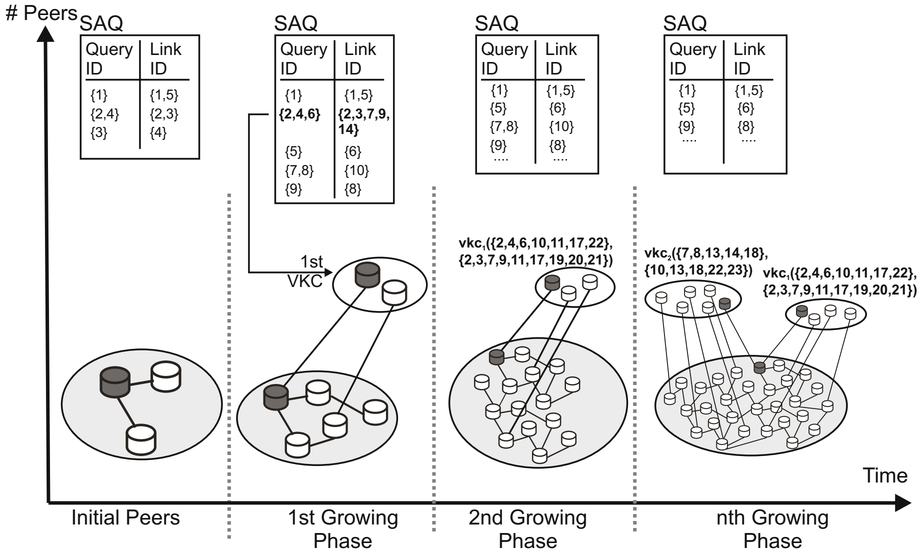


Fig. 3. Growing of Virtual Knowledge Communities

4.2 Growing of Virtual Knowledge Communities in a Peer-to-Peer Network

The growing of a VKC over some phases is visualized in figure 3.

In the upper half one can see the SAQ relation – depicted as a table – for each phase containing query and link IDs, which have been collected by the individual peers. One can consider the SAQ table as a database that has been realized on a well-known peer existing in the virtual network. This well-known peer is capable to receive so-called seldom asked queries from peers, which could not be assigned directly to a certain existing group. As time continues, the SAQ table grows with more links and queries, but also the number of peers having joined the virtual network. If a certain number of queries and links is transcended, representatives for Virtual Knowledge Communities can be identified (step 4 in section 4.1). In the given scenario in figure 3, a representative $vkc_1 \in VKC$ is computed in the first growing phase, after the virtual network has been initialized. The pertaining peer group is built afterwards, containing the respective peers that have previously pushed the query-link associations to the SAQ table. The generation of this new peer group will then be announced to all interested peers. Peers can pre-select the announcements for new Virtual Knowledge Communities through personalized filters that are placed on the same well-known peer. Once a VKC is created the respective representative for it is removed from the SAQ ($SAQ \setminus vkc_x$).

4.3 Distributed Retrieval System

Figure 4 demonstrates how a single query is being processed by ISKODOR. The user initiates the process by sending out a query-request. This request is then being sent to three different working units.

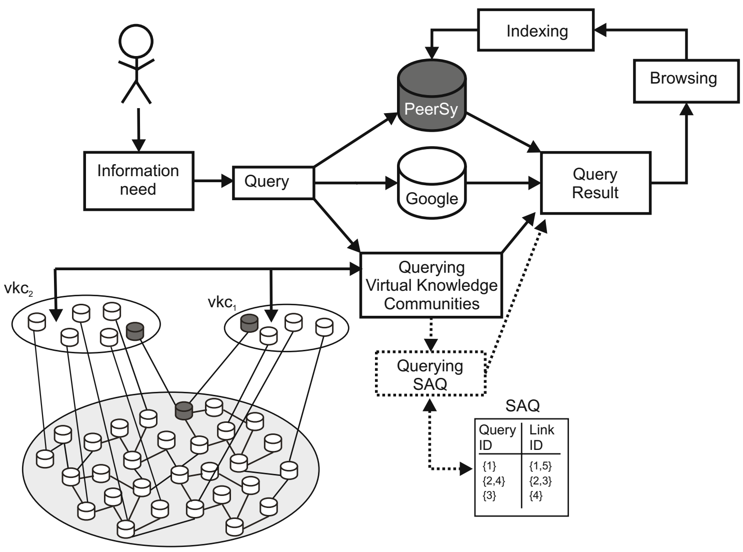


Fig. 4. Architecture of ISKODOR

A common internet search engine, like Google, executes its general query-process. At the same time, the PeerSy database is activated, looking for memorized links that the user has used formerly. In parallel to those two processes, the query is being sent out to the peer network. The query is being matched to the representative $vkc_x \in VKC$ of the different VKCs, starting with those groups, the user is already member of and then extending the matching onto the rest of the groups. There are two possibilities to consider now: (1) In case there is a match and the user/peer is already part of this group, he simply gets the appropriate links to his query. If the user/peer is not part of the group, the group itself can decide on how the peer could now join the group and on how many and what kind of links the user gets an inside on. This way the privacy of the group is assured and the group itself can decide on what kind of protection is needed. (2) The other case occurs if no VKC exists to this specific query and all matches with the representative were unsuccessful. In this case the query is being matched to the SAQ-list to find out if queries and links exist, that were just too few in number to create a Virtual Knowledge Community. If this matching has been successful, the query is being inserted to the corresponding queries and links. At this point the growing of the queries for this field of interest can even lead to the creation of a new peer group based on the queries and links concerned. If no match was found, the query is not being inserted into the SAQ-list. The network cannot satisfy the query-request and no corresponding links are sent back to the initiator.

After these processes are being run in parallel, the links from the three sources are being represented to the user. The user can now navigate through the hits and put relevant links into the PeerSy database. Once a new query-link entity has been inserted that had been found by an internet searching machine, a matching with the representative of the peer groups is initiated as described above. If a matching is found, the corresponding peer has the possibility to join this group. If no match is found, the query-link association is being inserted into the SAQ-list.

4.4 Design Improvements

This section discusses some design improvements of the overall ISKODOR system, which we will take into further consideration during the realization of the system.

The realization of the SAQ table as a single database is intuitive, but breaks the original notion of the peer-to-peer paradigm to abandon from having any kind of central server. If the SAQ server failed, the entire architecture as elucidated in figure 4 would fail. Moreover, the SAQ table may grow to a very large and unmanageable table after the nth growing phase. A possible solution we are currently analyzing is the implementation of the SAQ table as a distributed hash table (DHT). Likewise to conventional hash tables, a DHT is a data structure that maps "keys" to "values". The difference is that DHTs can be distributed over several nodes within a network. A concrete implementation of such a DHT is realized in the Content-Addressable Network (CAN) model by Ratnasamy et al. [Ratnasamy et al., 2001]. In this model, all peers are arranged in a (logical) d-dimensional Cartesian coordinate space. The entire coordinate space is dynamically partitioned into so-called zones among all the nodes in the system. Each node is dedicated as the owner of exactly one zone. The partition into zones is utilized to conduct requests (insert, lookup, or delete) to key/value pairs. Each key is mapped onto a point in the coordinate space through a common hash function. This point does also correspond to a distinct zone that is maintained by a peer. Following the CAN model the value of this key can be inserted or retrieved in the hash table of this peer. If this zone is not maintained by the requested node, the request is routed through a range of intermediate nodes towards the nodes that contains the key in his local hash table. To do so, each node additionally maintains a routing table that contains of a number of adjacent nodes in the table. The topology of a CAN-based system is not fixed, new nodes can be inserted, existing nodes can be deleted and so on. For more information about the algorithms for altering the topology see [Ratnasamy et al., 2001].

In order to improve our approach we are about to adopt the CAN model to store seldom asked queries in a decentral manner. Query-link associations are thereby represented by key/value pairs. During the initial phase, the SAQ table still remains on an initial peer, which is accessible through a well-known address. If a VKC is created, the initial table is splitted into two equally sized tables: one table remains on the initial peer, while the other table is migrated to the respective rendezvous peer that does represent and manage the VKC. In the following phases, the procedure of splitting and migrating a given SAQ table is conducted each time a VKC is identified within a SAQ table on a distinct rendezvous peer. Each rendezvous will also maintain a routing table consisting of adjacent rendezvous peer. This table will be updated each time a new SAQ table has been created. A query to a SAQ table can start from an arbitrary rendezvous peer. If the query cannot be matched in the local SAQ table, the query will be routed through all adjacent rendezvous peers.

In this regard, one essential question that has to be faced concerns the scalability of the presented model. According to the CAN model each peer has to maintain only $2d$ adjacent peers. Note that this number, in contrast to other routing strategies, is independent from the overall number of peers in the system. Thus, the number of nodes may grow without increasing the size of a routing table. One can conclude that the CAN model but also our adoption of CAN to our approach does scale even for huge numbers of new peers.

5 Related Work

In the following section some related systems are presented. Basically we refer to other studies which outline different methods to memorize and share bookmarks and to form groups based on interest. Particularly, we portray three systems that cover these areas of interest.

WebView (cf. [Cockburn et al., 1999]) is a prototype designed to improve the efficiency and usability of page revisitation. It does this by integrating many revisitation capabilities into a single display space, an add-on window that interacts with unaltered versions of Netscape Navigator. Whenever the user visits a page in Netscape, WebView keeps track of the page and applies a tree-like structure to the stored page-links. In contrast to PeerSy, this tool keeps track of every page that has been visited and it does not link the query that led to the page to the appropriate bookmark. Furthermore, this system does not support the collaborative aspect of sharing bookmarks between group members or the creation of groups of interest.

The system **CIRE (Collaborative Information Retrieval Environment)** (cf. [Romano et al., 1999]) is dedicated to support collaborative information seeking and retrieving. It constitutes the implementation of an integrated knowledge creation environment in which IR (Information Retrieval) and GSS (Group Support Systems) are combined to provide integrated group support for all tasks required for teams to work together, including information retrieval. The difference to ISKODOR appears to be the client-server architecture the system is build on. The ISKODOR architecture exploits both the extensive distributed resources available at the peers in addition to a centralized repository of the SAQ-list. This minimizes the role of centralized resources for low cost and scaling.

YouSearch (cf. [Bawa et al., 2003]) is a distributed (peer-to-peer) search application for personal Web servers operating within a shared context. It supports the aggregation of peers into overlapping (user defined) groups and the search over specific groups. The hybrid peer-to-peer architecture is augmented with a light-weight centralized component. In comparison to ISKODOR, YouSearch does not provide bookmark-sharing, but more or less file-sharing which is supported by search mechanisms. Another difference is that groups are formed via manual user action only and the system does not conduct any proposals (e.g. directly approaches the peers concerned to recommend a group creation) to support and enhance this process.

6 Conclusion

In this paper we presented our notion of Virtual Knowledge Communities based on peer-to-peer networks enabling users to share knowledge about web content on the basis of personal search memories. As future work we see the transformation of the presented concepts towards a prototype based on the Java platform and on the JXTA framework for peer-to-peer architectures. We intend to evaluate both the concepts and the prototype in a project made up of lawyers, who aim to share query-link associations for wordings of a law or annotations of convictions.

References

[Barkai, 2002] Barkai, D., editor (2002). *Peer-to-Peer Computing. Technologies for sharing and collaboration on the Net.* Intel Press.

[Bawa et al., 2003] Bawa, M., Bayardo, R. J., Rajagopalan, S., and Shekita, E. (2003). Make it fresh, make it quick – searching a network of personal webservers. In *Proc. ACM in Budapest, Hungary.*

[Cockburn et al., 1999] Cockburn, A., Greenberg, S., McKenzie, B., Smith, M., and Kaasten, S. (1999). Webview: A graphical aid for revisiting web pages. In *Proc. of the OZCHI'99 Austalian Conference on Human Computer Interaction.*

[Jain and Dubes, 1988] Jain, A. and Dubes, R. C., editors (1988). *Algorithms for Clustering Data.* Prentice Hall, pub-prentice:adr.

[Jones et al., 2001] Jones, W., Bruce, H., and Dumais, S. (2001). Keeping found things found on the web. In *Proc. of the 10th Int. Conf. on Information and Knowledge Management.*

[Nonaka, 1991] Nonaka, I. (1991). The knowledge creating company. *Harvard Business Review*, pages 96–104.

[Ratnasamy et al., 2001] Ratnasamy, S., Francis P., Handley, M., Karp, R., and Shenker, S. (2001). A Scalable Content Addressable Network. In *Proceedings of ACM SIGCOMM 2001.*

[Romano et al., 1999] Romano, N. C. J., Roussinov, D., Nunamaker, J. F. J., and Chen, H. (1999). Collaborative information retrieval environment: Integration of information retrieval with group support systems. In *Proc. of the 32nd Hawaii Int. Conf. on System Science 1999*.

[Stanoevska-Slabeva et al., 1998] Stanoevska-Slabeva, K., Hombrecher, A., Handschuh, S., and Schmid, B. (1998). Efficient information retrieval: Tools for knowledge management. In *Proc. the 2nd Int. Conf. on Practical Aspects of Knowledge Management (PAKM98)*, pages 23.1–23.6, Basel (Switzerland).

[Sun, 2003] Sun Microsystems Corp. (2003). Jxta v2.0 protocols specification. URL: http://spec.jxta.org/v2.0/.

The Personalized, Collaborative Digital Library Environment CYCLADES and Its Collections Management

Leonardo Candela and Umberto Straccia

I.S.T.I. – C.N.R.
Via G. Moruzzi, 1 I-56124 Pisa (PI) ITALY
{Leonardo.Candela,Umberto.Straccia}@isti.cnr.it

Abstract. Usually, a Digital Library (DL) is an information resource where users may submit queries to satisfy their daily information need. The CYCLADES system envisages a DL additionally as a personalized collaborative working and meeting space of people sharing common interests, where users (i) may organize the information space according to their own subjective view; (ii) may build communities, (iii) may become aware of each other, (iv) may exchange information and knowledge with other users, and (v) may get recommendations based on preference patterns of users. In this paper, we describe the CYCLADES system, show how users may define their own collections of records in terms of un-materialized views over the information space and how the system manages them. In particular, we show how the system automatically detects the archives where to search in, which are relevant to each user defined collection.

1 Introduction

Digital Libraries (DLs) [11] will play an important role not merely in terms of the information provided, but in terms of the *services* they provide to the information society. Informally, DLs can be defined as consisting of collections of information which have associated services delivered to user communities using a variety of technologies. The collections of information can be scientific, business or personal data, and can be represented as digital text, image, audio, video, or other media. Even though DLs have evolved rapidly over the past decade, typically, DLs still are limited to provide a search facility to the digital society at large. As DLs become more commonplace and the range of information they provide increases, users will expect more and more sophisticated services from their DLs. There is a need for DLs to move from being passive with little adaptation to their users, to being more proactive, or *personalized*, in offering and tailoring information for individual users. The requirement of a personalized search 'assistant' in the context of DLs is already known and, to date, some DLs provide related, though simplified, search functionality (see *e.g.* [3,7,8,9,10,13,14,17]). Informally, these DLs may fall in the so-called category of *alerting services*, *i.e.* services that notify a user (by sending an e-mail), with a list of references to new documents deemed as relevant. But, *searching* is just one aspect that should be addressed. Another orthogonal aspect of personalization concerns *information organization*, *i.e.* to support the users' interest in being able to organize the information space they are accessing according to *their own subjective perspective* (see *e.g.* [7,9]). Additionally, very seldom[1], a DL is also considered as a *collaborative meeting place* of people sharing common interests. Indeed, a DL may be viewed as a *common working place* where users may become aware of each other, open communication channels, and exchange information and knowledge with each other or with experts. In fact, usually users and/or communities access a DL in search of some information. This means that it is quite possible that users may have overlapping interests if the information available in a DL matches their expectations, backgrounds, or motivations. Such users might well profit from each other's knowledge by sharing opinions or experiences or offering advice. Some users might enter into long-term relationships and eventually evolve into a community if only they were to become aware of each other.

CYCLADES[2] is a DL environment supporting collaboration and personalization at various level, where users and communities may search, share and organize their information space according to their own personal view. While an extensive presentation of the CYCLADES system and its algorithm for filtering and recommendation has been given elsewhere [16], in this paper we will focus on how users may tailor the information space according to their subjective view. In particular, we will address the notion of *personalized (virtual) collections*. These are user defined un-materialized views over very heterogeneous information space, consisting of the archives adhering to the *Open Archives Initiative*[3] (OAI), available within CYCLADES. The main purpose of these personalized collections is to restrict the information space during the user's search task. To this purpose CYCLADES provides techniques of automated source selection [5,12] to automatically detect the archives relevant to a view.

The outline of the paper is as follows. In the next section we will recall the main features of CYCLADES, while in Section 3 we will address the management of personalized collections in CYCLADES and report some preliminary experimental results on the automated source selection procedure, which is at the core of the personalized collection management. Section 4 concludes the paper.

2 CYCLADES: A Personalized and Collaborative DL

The objective of CYCLADES is to provide an integrated environment for users and groups of users (communities) that want to use, in a highly personalized and flexible way, 'open archives', *i.e.* electronic archives of documents compliant with the OAI standard. Informally, the OAI is an initiative between several Digital Archives in order to provide interoperability among them. In particular, the OAI defines an easy-to-implement gathering protocol over HTTP, which give *data providers* (the individual archives) the possibility to make the documents' metadata in their archives externally available. This external availability of the metadata records then makes it possible for *service providers* to build higher levels of functionality. To date, there is a wide range of archives available in terms of its content, *i.e.* the family of OAI compliant archives is multidisciplinary

[1] [7] is an exception.

[2] `http://www.ercim.org/cyclades`

[3] www.openarchives.org.

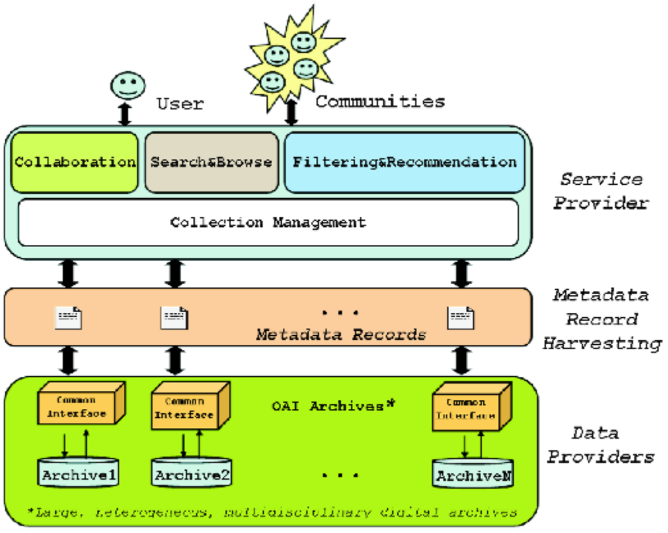
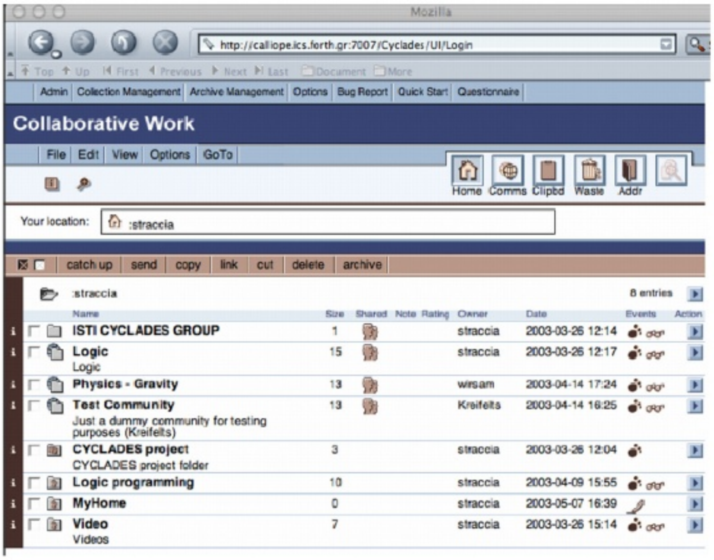


Fig. 1. Logical view of CYCLADES functionality.

in content. Under the above definition, CYCLADES *is an OAI service provider* (see Figure 1.) and provides functionality for (i) advanced search in *large, heterogeneous, multidisciplinary digital archives*; (ii) collaboration; (iii) filtering; (iv) recommendation; and (v) the management of records grouped into *collections*. Worth mentioning, the main principle underlying CYCLADES is the *folder paradigm* (see Figure 2). That is, users and communities of users may organize the information space into their own folder hierarchy, as *e.g.* may be done with directories in operating systems, bookmark folders in Web browser and folders in e-mail programs. A folder becomes a holder of information items, which are usually semantically related and, thus, implicitly determines what the folder's topic is about. Therefore, rather than speaking about a user profile, we will deal with a *folder profile*, *i.e.* a representation of what a folder is *about*. As a consequence, the user's set of folder profiles represents the set of topics the user is interested in and, thus, the profile of a user consists of the set of profiles related to his folders.

Figure 2, shows the home (top level) folder of a user. It contains several sub-folders. Among them, there are some (shared) folders belonging to communities (created by someone) to which the user joined to, like the 'Physics-Gravity' folder (community), while others are private folders and have been created directly by the user, *e.g.* the 'Logic Programming' folder. These folders contain community or user collected OAI records relevant to some topics (*e.g.* gravity and logic programming, respectively). Figure 3 shows the content of a folder, in our case the 'Physics-Gravity' folder of the community of physicists. In it there are several other folders and metadata records. Some records have been rated (*e.g.* the 'Astronaut Protection . . . ' record) and some records have notes attached (*e.g.* 'The Lunar Scout . . . ' record). There is also a discussion forum. These functionality are only some of those pertaining to the collaborative support package. Note



Your location: :straccia

:straccia 8 entries

Name	Size	Shared	Note	Rating	Owner	Date	Events	Action
ISTI CYCLADES GROUP	1				straccia	2003-03-26 12:14		
Logic Logic	15				straccia	2003-03-26 12:17		
Physics - Gravity	13				wirsam	2003-04-14 17:24		
Test Community Just a dummy community for testing purposes (Kreifelts)	13				Kreifelts	2003-04-14 16:25		
CYCLADES project CYCLADES project folder	3				straccia	2003-03-26 12:04		
Logic programming	10				straccia	2003-04-09 15:55		
MyHome	0				straccia	2003-05-07 16:39		
Video Videos	7				straccia	2003-03-26 15:14		

Fig. 2. User interface: a user home folder.

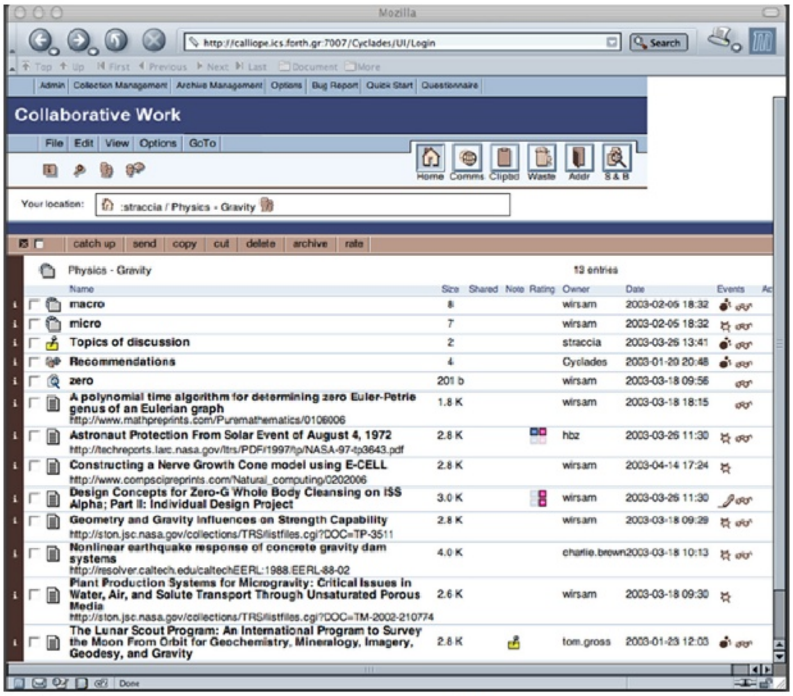
also that the CYCLADES system already provided some record, community, collection and user recommendations deemed by the system as relevant to this folder. Records retrieved after a search task may be stored in the folder by the user. This is the main way to populate folders with records gathered from CYCLADES information space (*i.e.* the OAI archives).

The architecture of the CYCLADES system is depicted in Figure 4. It should be noted that each box is a service accessible via the Web distributed over the Internet. The CYCLADES system, accessible through Web browsers, provides the user with different environments, according to the actions the user wants to perform. The functionality CYCLADES provides are developed by different services described next.

The *Collaborative Work Service* provides the folder-based environment for managing metadata records, queries, collections, external documents, received recommendations, ratings and annotations. Furthermore, it supports collaboration between CYCLADES users by way of folder sharing in communities, discussion forums and mutual awareness.

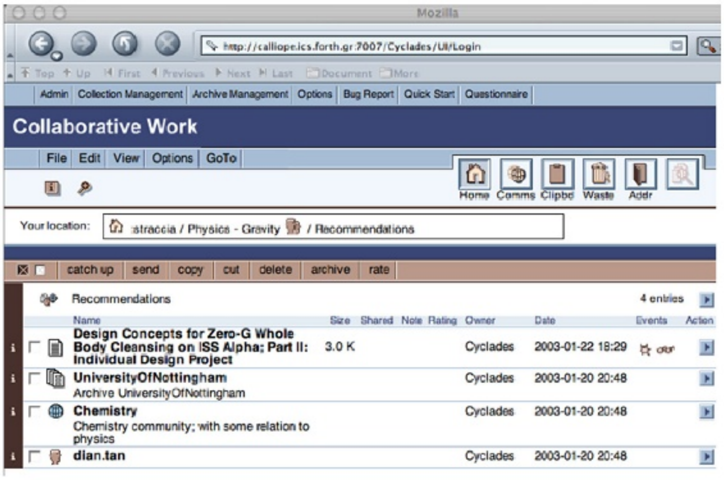
The *Search and Browse Service* supports the activity of searching records from the various collections, formulating and reusing queries associated to the folder by the user, and saving records to folders.

The *Access Service* is in charge of interfacing with the underlying metadata archives. In this project, only archives adhering to the OAI specification will be accounted for. However, the system is extensible to other kinds of archives by just modifying the Access


Mozilla
http://calliope.ics.forth.gr:7007/Cyclades/UI/Login
Search
Top Up First Previous Next Last Document More
Admin Collection Management Archive Management Options Bug Report Quick Start Questionnaire
Collaborative Work
File Edit View Options GoTo
Home Comms Clipbd Waste Addr S & B
Your location: :straccia / Physics - Gravity
catch up send copy cut delete archive rate
Done

Physics - Gravity — 13 entries

Name	Size	Shared	Note	Rating	Owner	Date
macro	8				wirsam	2003-02-05 18:32
micro	7				wirsam	2003-02-05 18:32
Topics of discussion	2				straccia	2003-03-26 13:41
Recommendations	4				Cyclades	2003-01-20 20:48
zero	201 b				wirsam	2003-03-18 09:56
A polynomial time algorithm for determining zero Euler-Petrie genus of an Eulerian graph http://www.mathpreprints.com/Purematematics/0106006	1.8 K				wirsam	2003-03-18 18:15
Astronaut Protection From Solar Event of August 4, 1972 http://techreports.larc.nasa.gov/ltrs/PDF/1997/tp/NASA-97-tp3643.pdf	2.8 K				hbz	2003-03-26 11:30
Constructing a Nerve Growth Cone model using E-CELL http://www.compscipreprints.com/Natural_computing/0202006	2.8 K				wirsam	2003-04-14 17:24
Design Concepts for Zero-G Whole Body Cleansing on ISS Alpha; Part II: Individual Design Project	3.0 K				wirsam	2003-03-26 11:30
Geometry and Gravity Influences on Strength Capability http://ston.jsc.nasa.gov/collections/TRS/listfiles.cgi?DOC=TP-3511	2.8 K				wirsam	2003-03-18 09:29
Nonlinear earthquake response of concrete gravity dam systems http://resolver.caltech.edu/caltechEERL:1988.EERL-88-02	4.0 K				charlie.brown	2003-03-18 10:13
Plant Production Systems for Microgravity: Critical Issues in Water, Air, and Solute Transport Through Unsaturated Porous Media http://ston.jsc.nasa.gov/collections/TRS/listfiles.cgi?DOC=TM-2002-210774	2.6 K				wirsam	2003-03-18 09:30
The Lunar Scout Program: An International Program to Survey the Moon From Orbit for Geochemistry, Mineralogy, Imagery, Geodesy, and Gravity	2.8 K				tom.gross	2003-01-23 12:03


Mozilla
http://calliope.ics.forth.gr:7007/Cyclades/UI/Login
Top Up First Previous Next Last Document More
Admin Collection Management Archive Management Options Bug Report Quick Start Questionnaire
Collaborative Work
File Edit View Options GoTo
Home Comms Clipbd Waste Addr
Your location: :straccia / Physics - Gravity / Recommendations
catch up send copy cut delete archive rate

Recommendations — 4 entries

Name	Size	Shared	Note	Rating	Owner	Date
Design Concepts for Zero-G Whole Body Cleansing on ISS Alpha; Part II: Individual Design Project	3.0 K				Cyclades	2003-01-22 18:29
UniversityOfNottingham Archive UniversityOfNottingham					Cyclades	2003-01-20 20:48
Chemistry Chemistry community; with some relation to physics					Cyclades	2003-01-20 20:48
dian.tan					Cyclades	2003-01-20 20:48

Fig. 3. User interface: folder content and recommendations.

Service only. A user may also ask CYCLADES to include newly OAI compliant archives as well.

The *Collection Service* manages personalized collections (*i.e.* their definition, creation, and update) and stores them, thus allowing a dynamic partitioning of the in-

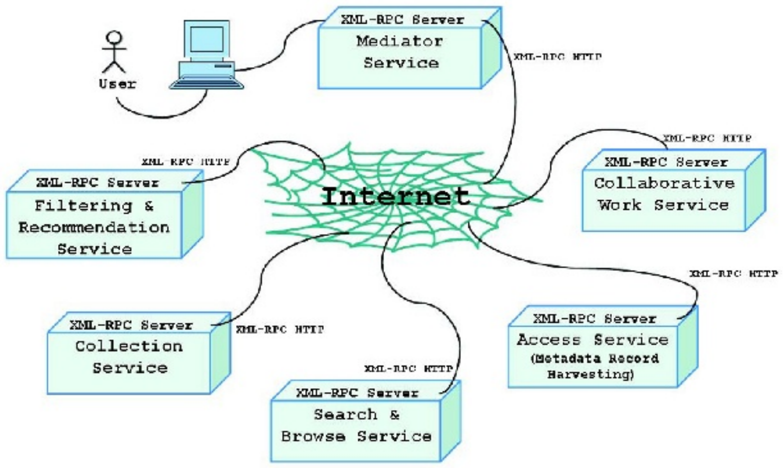


Fig. 4. Architecture.

formation space according to the users' interests, and making the individual archives transparent to the user.

The *Filtering and Recommendation Service* provides filtered search, recommendations of records, collections, users, and communities deemed relevant to the user's interests.

Finally, the *Mediator Service*, the main entry point to the CYCLADES system, acts as a registry for the other services, checks if a user is entitled to use the system, and ensures that the other services are only called after proper authentication.

The Collaborative Work Service, the Search and Browse Service, the Access Service, and the Collection Service provide their own user interfaces. The Mediator Service itself provides the registration and login interface, and a system administration interface (for assigning access rights, etc.). Additionally, the Mediator Service integrates the user interfaces of the other services, and makes sure that those services and their interfaces are called only for authorized users, and only via the Mediator Service.

3 Personalized Collection Management in CYCLADES

We present some details on the management of personalized collections within CYCLADES, in particular in the Collection Service. As already addressed, the Collection Service introduces a mechanism to support users and/or communities (in the following called *agent*) to define their own information space. Usually, a collection is meant to reflect a topic of interest of an agent, *e.g.* the collection of records about information retrieval. To facilitate an agent's personalized view over the information space, an agent may organize its own defined collections into an hierarchical order. A major distinction of the Collection Service is that collections are not materialized, but are rather *personalized (virtual) collections*, *i.e.* in database terms, un-materialized views over the information

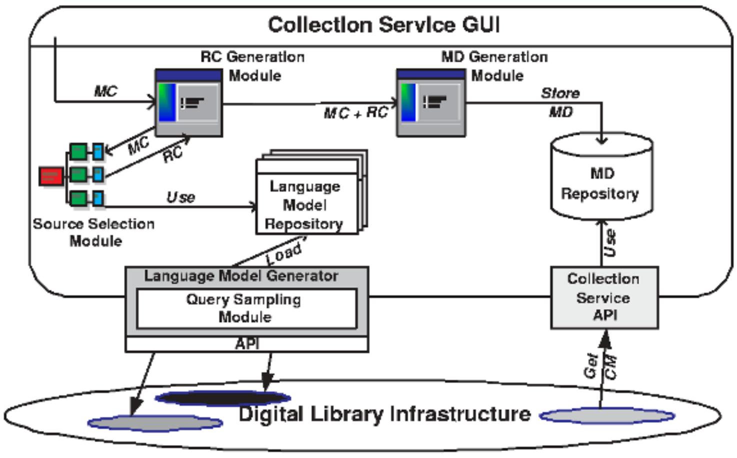
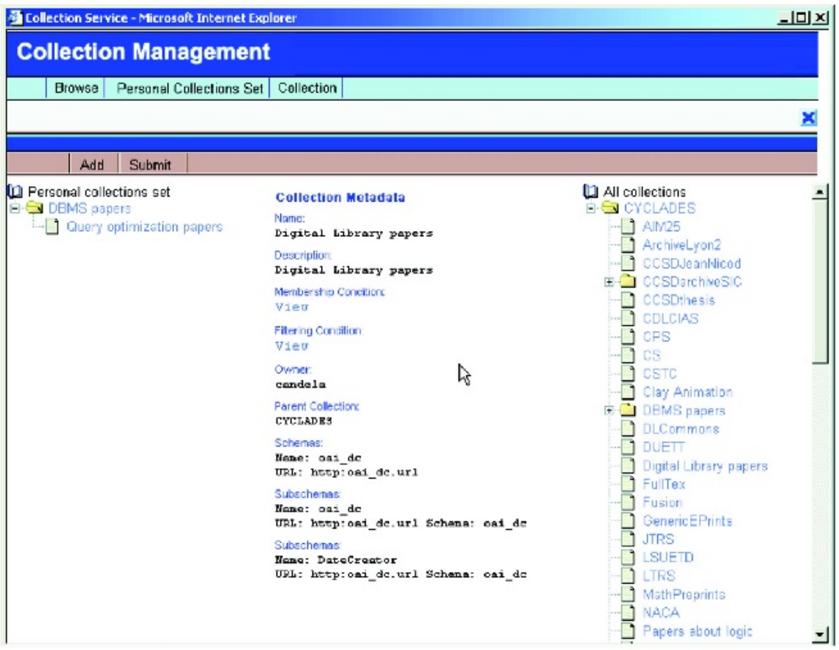


Fig. 5. The CYCLADES Collection Service Logical Architecture.

space. That is, *e.g.* a personal collection of an agent, whose aim is to collect records about 'information retrieval' is not a database table containing records, which are about information retrieval, but rather is a specification of the conditions that a record should satisfy in order to belong to this collection. Figure 5 shows the logical architecture of the Collection Service. The agent defines a collection using the *Collection Definition Language* (CDL). By means of the CDL, it is possible to define the *Membership Condition* (MC) of a collection that the metadata records should satisfy in order to be part of the defining collection. To reduce the information space effectively, the Collection Service uses then the MC of a collection to automatically determine the most relevant resources in which to search for records meeting the MC. This is done by relying on techniques known as *automatic source selection* (see, *e.g.* [2,15,12]). This set of determined OAI archives is then added to the MC and the resulting description is called *Retrieval Condition* (RC). The MC and the RC, together with some other collection related data forms the *Collection Metadata* (MD), which is then stored into the MD repository. While the MC is modified by an agent only, its relative RC is modified by the Collection Service whenever appropriate. So, for instance, if a new resource is added to CYCLADES then the RCs are re-computed and the new resource may become part of the MCs. This allows the Collection Service to deal with a dynamic number of OAI compliant archives, whose content may vary over time. That is, the Collection Service follows the dynamism of the underlying information space. As the Collection Service does not have the archives (which are in the Access Service), the Collection Service automatically computes an approximation of the content of each OAI compliant archive registered in CYCLADES, to support the function of automated source selection. This data is stored in the Language Model Repository. The approximation is computed by relying on the so-called *query-based sampling method* [6].

Figure 6 shows an excerpt of the Collection Service user interface. In this particular case, on the right column the set of all collections created within CYCLADES (by users, communities or automatically by CYCLADES itself) is shown. The left column shows the collections pertaining to a particular user, while the middle column shows the specification of a collection. A user may anytime define new collections, add collections to its particular collection, etc. In the following, we will address more specifically some aspects of the Collection Service. Namely, the collection definition language, the query-based sampling method, the automated source selection method and some preliminary test results.



Collection Metadata

Name:
Digital Library papers

Description:
Digital Library papers

Membership Condition:
View

Filtering Condition:
View

Owner:
candela

Parent Collection:
CYCLADES

Schemas:
Name: oai_dc
URL: http:oai_dc.url

Subschemas:
Name: oai_dc
URL: http:oai_dc.url Schema: oai_dc

Subschemas:
Name: DateCreator
URL: http:oai_dc.url Schema: oai_dc

Fig. 6. CS Graphical User Interface for select the Personal Collections Set.

Collection definition language. The Collection Service allows users to specify their own information needs via a declarative collection definition language. Each definition specifies a list of conditions that a record has to satisfy in order to belong to the set. The definitions are soft in the sense that each record may satisfy them to some degree in the unit interval $[0, 1]$. Therefore, a collection may be seen as a *fuzzy set* [19] of records. The language is simple, but expressive and quite usable. Below, we present the CDL syntax in Backus Naur Form (BNF).

```
query       ::= condition* [, (archiveList)]
condition   ::= ([weight,] field, predicate, value)
weight      ::= + | - | 1..1000
field       ::= [schemaName":"]attributeName
predicate   ::= cw | < | <= | >= | > | = | ! =
archiveList ::= archiveName | archiveName, archiveList
```

Roughly, this is an ALTAVISTA-style language where a query is a set of conditions, which are either optional, *mandatory* (+) or *prohibitive* (-). In addition, it allows for weighting of optional conditions. With respect to the structure of metadata records it assumes that they have a one-level structure[4]. It allows the use of a name space (*e.g.* `schemaName`). The set of predicates supported is composed by the classical comparison operators ($<$, $<=$, $>=$, $>$, $=$ and $!=$) plus the `cw` operator used to specify a condition on the content (aboutness) of a text field, *e.g.* `(description, cw, library)` stands for "the value of the attribute `description` is relevant to the term `library`".

Automated source selection in the Collection Service. In the CYCLADES system, a query is issued from the Search and Browse Service. As specified early, from the design choice of CYCLADES a global index for all OAI compliant archives does not exists. Therefore, query evaluation is performed by dispatching a query to all OAI compliant archives. A feature of the Search and Browse Service is that the query may be accompanied with the specification of a collection[5]. In this latter case, the query has to be understood as a refinement of the information space and records are searched only in OAI archives relevant to the collection specification. To this end, the RC, automatically built by the Collection Service from the membership condition, is coupled with the query and the result of this combination is sent to the Access Service to compute the result. The RC consists of the MC plus a set of automatically determined archives most relevant to the conditions specified in the MC only. The language chosen for specifying the MC is similar to the query language supported by the Search and Browse Service. Therefore, no significant query reformulations are necessary to build the RC, except the addition of the determined relevant archives.

The computation of the RC from the MC needs two steps:

– the computation of an approximation of the content of each OAI compliant archive registered in CYCLADES;
– the selection of those archives deemed as most relevant to the collection definition, relying on the approximations of the archives' content.

The first step is done periodically and each time a new archive is added to CYCLADES, while the second step is done immediately after a personalized (virtual) collection has been defined by an agent. RCs are updated periodically as well. In the following, we will address these two steps further. In order to select the relevant information sources for a query the Collection Service has to have a sort of knowledge about their content. For each archive, we build a simple language model of it (a list of terms with their term weight information). We rely on the so-called *query-based sampling method* [6], which has been proposed for automatically acquiring statistical information about the content of an information source. A major feature is that it requires only that an information source provides a query facility and access to the documents, that are in the result of a query. Informally, the method is an iteration of the following steps 2 and 3: (1) issue a random query to the Access Service (as start-up); (2) select the top-k documents;

[4] The assumption about the one-level metadata record structure can be removed using the attribute name path instead of the attribute name.

[5] More precisely, a list of collections is allowed.

Table 1. Sampling Algorithm.

```
1: query = generateInitialTrainingQuery();
2: resultSet = run(query);
3: if(|resultSet| $< L_{tr}$){
4:     go to 1;
5: }else{
6:     updateResourceDescription(resultSet);
7:     if(NOT stoppingCriteria()){
8:         query = generateTrainingQuery();
9:         resultSet = run(query);
10:        go to 6;
11:    }
12: }
```

and (3) build m new queries from n randomly selected terms within the top-k ranked documents. The iteration continues until a stop criterion is satisfied. While [6] worked with plain text, in our context we have Dublin Core metadata records. Table 1 shows the detailed sampling algorithm, customized to the case where text databases have multiple text attributes (*e.g.* bibliographic records, similar to [18]).

This algorithm uses a set of functions:

generateInitialTrainingQuery() it generates the *start* training query. In order to generate a query we need: (a) a set of attributes among which we randomly choose the ones to be used to build the initial condition; and (b) a set of terms among which we randomly choose the ones to fill-in the attributes values. For each selected attribute we randomly select 1 to max_t distinct terms and for each (attribute, value) pair we choose an operator to relate attribute and term into the condition. In the CYCLADES Collection Service prototype we have adopted the following decision: (a) concerning the terms, we use the set of terms that characterize the second and the third level of Dewey Decimal Classification [1], (b) concerning the attributes, we have used Dublin Core[6] fields, (c) $max_t = 4$ and (d) the operator to use is always the `cw` operator.

updateResourceDescritpion() it updates the set of records that represents the resource description. Note that a query must return at least L_{tr} records before the records collected (the top L_{tr}) can be added to the resource description record set. This minimum result size is required because query returning small results do not capture source content well. In our prototype we have used $L_{tr} = 4$ as proposed in [18], this is just another configuration aspect.

stoppingCriteria() it evaluates if the stopping criteria was reached. Callan and Connell [6] stop the sampling after examining 500 documents, a stopping criteria chosen empirically observing that increasing the number of documents examined does not improve significantly the language model.

generateTrainingQuery() it generates the *next* training query. Training queries are generated as follow:

[6] `http://dublincore.org/`

1. randomly select a record R from the resource description record set;
2. randomly select a set of attributes of R to be used in the training query;
3. for each attribute to be included in the training query, construct a predicate on it by randomly select 1 to max_t distinct terms (stopwords are discarded) from the corresponding attribute value, by using the `cw` operator.

At the end of this process a sample of records of the information source is acquired. This set is called resource description and the Collection Service uses it to build the language model of the archive.

We conducted some preliminary experiments to evaluate the quality of the computed archive approximations. We have considered two bibliographic information sources. A very small and homogeneous information source (*Archive 1*, 1616 records, 13576 unique terms after stopwords removal, papers about computer science published by the same authority) and a more large and heterogeneous information source (*Archive 2*, 16721 records, 79047 unique terms after stopwords removing, papers published by different authorities). Note that OAI compliant archives are characterized to be very small in terms of numbers of records ($\leq$ 2000 records) except some few exceptions, like arXiv[7] ($\approx$ 270000 records).

The experimental method was based on comparing the learned resource description of an information source with the real resource description for that information source. Resource descriptions can be represented using two information, a vocabulary V of the set of terms appearing in the information source records and a frequency information for each vocabulary term: the number of documents containing the terms, called *document frequency* (*df*).

In accordance with [6], we have used two metrics to evaluate the quality of resource descriptions acquired by sampling: the *ctf ratio* (CTF) to measure the correspondence between the learned vocabulary (V') and the real vocabulary (V) and the *Spearman Rank Correlation Coefficient* (SRCC) to measure the correspondence between the learned and the real document frequency information. This metrics are calculated using Equation (1) and (2) where ctf_i is the number of times term i occurs in the resource description of information source i, d_i is the rank difference of a common term $t_i \in V' \cap V$. The two term rankings are based on the learned and the actual document frequency df_i. n is the total number of common terms.

$$\mathrm{CTF} = \frac{\sum_{i \in V'} ctf_i}{\sum_{i \in V} ctf_i} \tag{1}$$

$$\mathrm{SRCC} = 1 - \frac{6}{n^3 - n} \sum_{t_i \in V' \cap V} {d_i}^2 \tag{2}$$

Five trials were conducted for each information source and for each trial a resource description of 500 records has been acquired to illustrate the behavior of the measures

[7] http://arxiv.org

above. The results reported here are the average of the results of each trial. Figures 7–10 show respectively the CTF and the SRCC metrics calculated for some Dublin Core attribute. On the x-axis, we varied the number of acquired records.

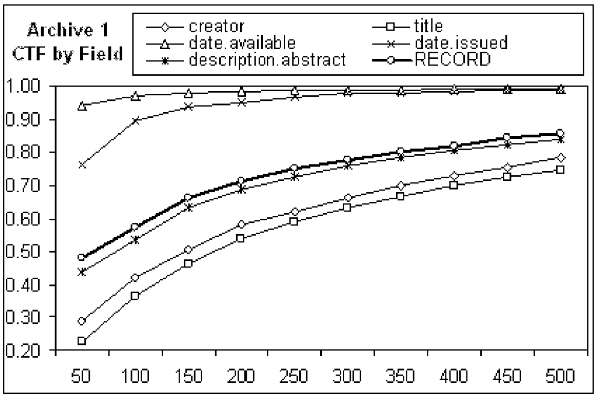


Fig. 7. Archive 1: CTF.

By observing the CTF graphics we can note that the language model acquired for the first archive is better then the one acquired for the second. Moreover we can note that the language model acquired for a field has different characteristics than the one acquired for other fields. The reasons for this behaviour are twofold: *Archive 2* contains more records and is more heterogeneous than *Archive 1* and some attributes, *e.g.* `creator`, are more heterogeneous than others, *e.g.* `date`. The more heterogeneous the values of an attribute are, the more difficult it is to approximate it.

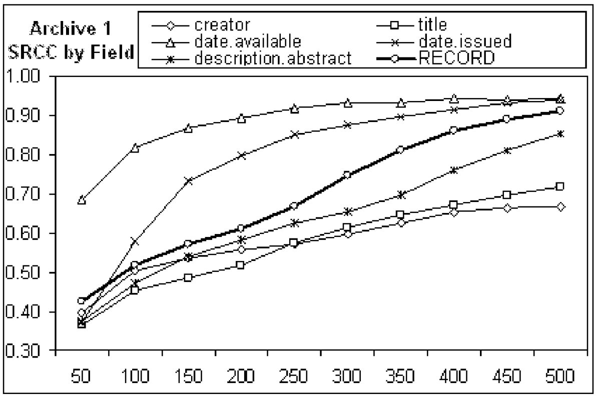


Fig. 8. Archive 1: SRCC.

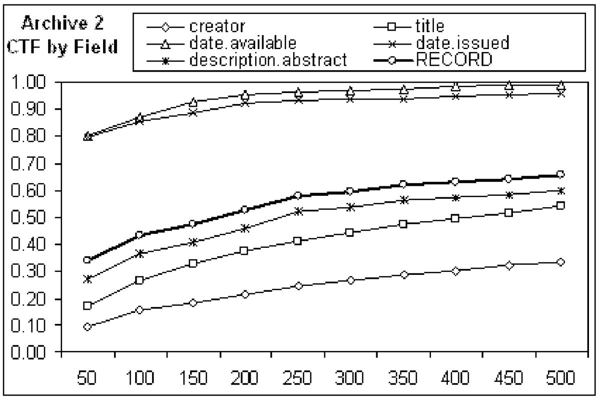


Fig. 9. Archive 2: CTF.

By observing the SRCC graphics we can note that the *quality* of the language model acquired via sampling is high, considering the record as a plain text we can found values greater that 80% (see RECORD line).

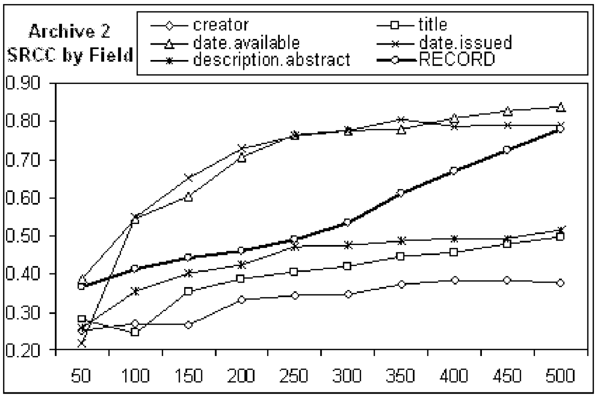


Fig. 10. Archive 2: SRCC.

Let us now deal with the automated source selection problem. Source selection is the problem of selecting from a large set of accessible information sources the ones relevant to a given query. In our case the query is the MC, *i.e.* the collection characterization criteria, while the selected information sources are used in the generation of the RC.

Our approach to automated source selection extends the CORI scheme [4] to the case of OAI-compliant archives, *i.e.* where records in the Dublin Core format are provided. Informally, given a membership condition MC, to each archive IS_i we associate a goodness value $G(MC, IS_i)$ and then select the top-k ranked archives. Formally, let MC be a set of conditions $c_j = (w_i, a_i, o_i, v_i)$ where:

- w_i is the *weight* of this condition, where $w_k \in [1..1000] \cup \{+, -\}$. $+$ means that the condition must be fulfilled, $-$ means that the condition must not be fulfilled (the boolean NOT);
- a_i is the attribute of the Dublin Core record involved in the condition;
- o_i is the operator, *e.g.* `<=`, `=`, `cw`, etc.;
- v_i is the term.

For instance, `(+,author,cw,''Straccia'')(+,subject,cw,''Cyclades'')` denotes the (fuzzy) set of records having author "Straccia" and "subject" is related to "Cyclades".

The $Goodness$ score $G(MC, IS_i)$ for an information source IS_i and membership condition MC is defined as follow:

$$G(MC, IS_i) = \begin{cases} 0 & \text{if } \exists k \in [1..|MC|], s.t., w_k \in \{+, -\} \wedge p(c_k|IS_i) = 0 \\ \dfrac{\sum_{k=1}^{|MC|} p(c_k|IS_i)}{|MC|} & \text{otherwise} \end{cases}$$

where the "belief" $p(c_k|IS_i)$ in IS_i, for condition c_k is defined as

$$p(c_k|IS_i) = \begin{cases} T_{i,k} \cdot I_k \cdot w_k & \text{if } w_k \in [1..1000] \\ T_{i,k} \cdot I_k & \text{if } w_k = \text{"}+\text{" or } w_k = \text{"}-\text{"} \end{cases}$$

$$T_{i,k} = \frac{df_{i,k}}{df_{i,k} + 50 + 150 \cdot \dfrac{cw_{i,k}}{\overline{cw_k}}}$$

$$I_k = \frac{\log\left(\frac{|S|+0.5}{cf_k}\right)}{\log\left(|S| + 1.0\right)}$$

where:

$df_{i,k}$	is the number of records in the approximation of IS_i satisfying c_k;
$cw_{i,k}$	is the number of terms in attribute a_k of records in the approximation of IS_i;
$\overline{cw_k}$	is the mean value of $cw_{\cdot,k}$ over the approximation of IS_i;
cf_k	is the number of approximated information sources that satisfy c_k;
$\lvert S\rvert$	is the number of the information sources.

We carried out some preliminary experiments to evaluate the efficiency and effectiveness of the above source selection method. Indeed, we generated randomly 200 collections using Dublin Core fields. The collections generated are of two kinds: 100 collections (T1) are generated using a combination of conditions on `description` and `title` fields, 100 collections (T2) are generated using a combination of conditions on all fields of the Dublin Core schema. Given a collection definition MC_i and the relative RC_i obtained after source selection, $Precision_i$ is defined as the quantity

$$Precision_i = \frac{|ret(RC_i) \cap ret(MC_i)|}{|ret(RC_i)|}$$

and $Recall_i$ is defined as the quantity

$$Recall_i = \frac{|ret(RC_i) \cap ret(MC_i)|}{|ret(MC_i)|} \; .$$

$ret(MC_i)$ is the set of records retrieved by submitting the MC_i query to CYCLADES consisting of 50 OAI archives (taking the top-100 records). $ret(MC_i)$ is considered as the set of records effectively to be retrieved. $ret(RC_i)$ is the set of records retrieved by submitting RC_i as query (restricting the set of archives in which to search for records). Precision and recall measure how effective the source selection method is with respect to the original query MC_i, which considered all information sources stored in CYCLADES. For each pair (MC_i, RC_i), precision measures the conditional probability $\mathcal{P}(ret(MC_i)|ret(RC_i))$, while recall measures $\mathcal{P}(ret(RC_i)|ret(MC_i))$. For each pair (MC_i, RC_i) we have a precision/recall value $(Precision_i, Recall_i)$. These pairs of values have been partitioned into precision/recall levels and are summarized in Table 2. In it, each row/column pair (r, p), where r and p are intervals denoting respectively recall level and precision level, dictates the percentage of test pairs (MC_i, RC_i) such that $Recall_i \in r$ and $Precision_i \in p$. Furthermore, the right most column and the bottom row report the total amount w.r.t. a row and a column, respectively. For instance, from Table 2 we have that 27.5% of the test pairs (MC_i, RC_i) have recall and precision level in $[0.91, 1]$, while 96.66% of the tests have precision level in $[0.91, 1]$.

Table 2. Source selection: precision and recall.

		Precision										
		0.00 – 0.10	0.11 – 0.20	0.21 – 0.30	0.31 – 0.40	0.41 – 0.50	0.51 – 0.60	0.61 – 0.70	0.71 – 0.80	0.81 – 0.90	0.91 – 1.00	
	0.00 – 0.10	0.33%	0	0	0	0	0	0	0	0	8%	8.33%
	0.11 – 0.20	0	0	0	0	0	0.16%	0	0	0	5.83%	6%
R	0.21 – 0.30	0	0	0	0	0	0	0	0	0	5.83%	5.83%
e	0.31 – 0.40	0	0	0	0	0	0	0	0	0	7.5%	7.5%
c	0.41 – 0.50	0	0	0	0	0	0.16%	0	0	0.16%	12.16%	12.5%
a	0.51 – 0.60	0	0	0	0	0	0.16%	0	0	0	2.5%	2.66%
l	0.61 – 0.70	0	0	0	0	0	0	0.16%	0	0	8.66%	8.83%
l	0.71 – 0.80	0	0	0	0	0	0	0	0.5%	0.33%	8.83%	9.66%
	0.81 – 0.90	0	0	0	0	0	0	0	0	1.33%	9.83%	11.16%
	0.91 – 1.00	0	0	0	0	0	0	0	0	0	27.5%	27.5%
		0.33%	0	0	0	0	0.5%	0.16%	0.5%	1.83%	96.66%	

The RC effectively improves the performance. Table 3 shows a measure of this improvement. In particular, it compares the average query response times obtained by retrieving the set of documents matching the MC_i with those obtained using the RC_i.

In summary, Table 2 and Table 3 show that there is an high improvement in response time with little loss in the set of records retrieved after automatic source selection.

Table 3. Source selection: average response time.

	T1	T2	Average
MC	162874 ms	186909 ms	174892 ms
RC	48469 ms	52253 ms	50361 ms
Improvement in ms	114405 ms	134655 ms	124530 ms
Improvement in %	70.24%	72.04%	71.20%

4 Conclusions

Since the Web and the information contained in it, is growing rapidly, every day a huge amount of "new" information is electronically published and new Digital Libraries are available to satisfy the user information needs. In this paper, we described a Digital Library that is not only an information resource where users may submit queries to get what they are searching for, but also a personalized, collaborative working and meeting space in which the user functionality may be organized into four categories: users may (i) search for information; (ii) organize the information space (according to the folder and personalized collections paradigm); (iii) collaborate with other users sharing similar interests; and (iv) get recommendations. Particular attention has been paid to the notion of personalized collections, which are user defined un-materialized views of the information space, and how the system automatically determines the most relevant information sources related to the views.

Acknowledgements. This work is funded by the European Community in the context of the CYCLADES project IST-2000-25456, under the Information Societies Technology programme.

References

1. Dewey Decimal Classification. `http://www.oclc.org/dewey`.
2. Christoph Baumgarten. A probabilistic solution to the selection and fusion problem in distribute information retrieval. In *Proc. of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 246–256, Berkeley, CA USA, 1999.
3. K. Bollacker, S. Lawrence, and C. L. Giles. A system for automatic personalized tracking of scientific literature on the web. In *The 4th ACM Conference on Digital Libraries*, pages 105–113, New York, 1999. ACM Press.
4. J. P. Callan, Z. Lu, and W. Bruce Croft. Searching distributed collections with inference networks. In E. A. Fox, P. Ingwersen, and R. Fidel, editors, *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 21–28, Seattle, Washington, 1995. ACM Press.

5. Jamie Callan. Distributed information retrieval. In W.B. Croft, editor, *Advances in Information Retrieval*, pages 127–150. Kluwer Academic Publishers, Hingham,MA, USA, 2000.
6. Jamie Callan and Margaret Connell. Query-based sampling of text databases. *ACM Transactions on Information Systems*, 19(2):97–130, 2001.
7. M. Di Giacomo, D. Mahoney, J. Bollen, A. Monroy-Hernandez, and C.M. Ruiz Meraz. Mylibrary, a personalization service for digital library environments, 2001.
8. D. Faensen, L. Faulstich, H. Schweppe, A. Hinze, and A. Steidinger. Hermes: a notification service for digital libraries. In *ACM/IEEE Joint Conference on Digital Libraries*, pages 373–380, 2001.
9. L. Fernandez, J. A. Sanchez, and A. Garcia. Mibiblio: personal spaces in a digital library universe. In *ACM DL*, pages 232–233, 2000.
10. P.W. Foltz and S.T. Dumais. Personalized information delivery: an analysis of information filtering methods. *Communications of ACM*, 35(12):51–60, 1992.
11. Edward A. Fox and Gary Marchionini. Digital libraries: Introduction. *Communications of the ACM*, 44(5):30–32, 2001.
12. Norbert Fuhr. A decision-theoretic approach to database selection in networked IR. *ACM Transactions on Information Systems*, 3(17):229–249, 1999.
13. *Information Filtering Resources:* `http://www.enee.umd.edu/medlab/filter`, WWW.
14. A. Moukas. *Amalthaea*: Information discovery and filtering using a multiagent evolving ecosystem. In *Proceedings Practical Applications of Agents and Multiagent Technology*, London, GB, 1996.
15. Allison L. Powell, James C. French, and Jamie Callan. The impact of database selection on distributed searching. In *Proc. of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 232–239, Athens, Greece, 2000.
16. M. Elena Renda and Umberto Straccia. A personalized collaborative digital library environment. In *5th International Conference on Asian Digital Libraries (ICADL-02)*, number 2555 in Lecture Notes in Computer Science, pages 262–274, Singapore, Republic of Singapore, 2002. Springer-Verlag.
17. L.M. Rocha. Talkmine and the adaptive recommendation project. In *ACM DL*, pages 242–243, 1999.
18. Jian Xu, Yinyan Cao, Ee-Peng Lim, and Wee-Keong Ng. Database selection techniques for routing bibliographic queries. In *Proceedings of the third ACM conference on Digital Libraries*, pages 264–274. ACM Press, 1998.
19. L. A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.

Author Index